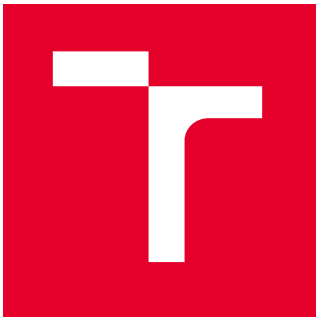


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

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

SEBELOKALIZACE A MAPOVÁNÍ VE VNITŘNÍM PROSTŘEDÍ NA BÁZI LIDAR SNÍMAČŮ

LIDAR-BASED INDOOR SELF LOCALIZATION AND MAPPING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jakub Minařík**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Adam Ligocki, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Kybernetika, automatizace a měření**

Ústav automatizace a měřicí techniky

Student: Bc. Jakub Minařík ***ID:*** 203294

Ročník: 2 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Sebelokalizace a mapování ve vnitřním prostředí na bázi LiDAR snímačů

POKYNY PRO VYPRACOVÁNÍ:

Práce se zabývá vytvořením systému schopného sebelokalizace a mapování (SLAM) pomocí 3D laserových dálkoměrů (LiDAR). Tento systém pak bude posléze nainstalován na existujícího robota a funkcionalita celé sestavy bude otestována ve vnitřních prostorách budovy Technická 12.

1. Proveďte rešerši existujících open-source SLAM projektů založených na 3D LiDARových datech, zaměřte se na projekty kompatibilní se systémem ROS.
2. Na základě konzultace s vedoucím, vyberte vhodný projekt a nastudujte jeho principy fungování a datové vstupy a výstupy.
3. Seznamte se s dostupných hardwarem a navrhněte jak HW, tak SW architekturu výsledného sytému.
4. Proveďte oživení projektu vybraného v bodě 2 a testujte jej na předpřipravených offline datech.
5. Přeneste systém na mobilního robota a otestujte jej na online datech. Srovnejte výsledky s bodem 4.
6. Diskutujte kvalitu výstupů Vámi realizovaného SLAM systému, tedy výstupní mapy a na ní založené lokalizace robota. Zvažte možnost použití systému ve vnějším prostředí.

DOPORUČENÁ LITERATURA:

THRUN, Sebastian, Wolfram BURGARD a Dieter FOX. Probabilistic robotics. Massachusetts: MIT Press, c2006. ISBN 02-622-0162-3.

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 17.5.2023

Vedoucí práce: Ing. Adam Ligocki, Ph.D.

doc. Ing. Petr Fiedler, Ph.D.
předseda rady studijního programu

ABSTRAKT

Tato práce seznamuje s problémem sebelokalizace a mapování (SLAM) se zaměřením na použití snímače 3D LiDAR. Nejprve je proveden úvod od samotného SLAMu a vysvětlen SLAM založený na grafu a reprezentace mapy pomocí dense SLAM. Popsány jsou dva nejběžnější algoritmy pro porovnání mračen bodů ICP a NDT. Po popsání teorie SLAMu je provedena rešerše existujících projektů řešící tento problém. Popsané projekty jsou open-source a většina z nich podporuje ROS systém. Z popsaných projektů je vybrán projekt Optimized SC-F-LOAM pro další práci. Je popsána jak funguje jeho odometrie FLOAM a její spojení s optimalizací a hledání smyček. Hledání smyček je prováděno pomocí deskriptoru ScanContext. Následně je proveden návrh pro zprovoznění projektu pro offline a online dat ve vnitřních prostorách. V posledních kapitolách je řešen postup oživení, ladění projektu a jsou vyhodnoceny výsledku daného projektu pro použití ve vnitřních prostorech.

KLÍČOVÁ SLOVA

SLAM, 3D LiDAR, odometrie LiDARu, mračna bodů, ROS, lokalizace, mapování, Optimized SC-F-LOAM, F-LOAM, vnitřní SLAM, FLOAM, SC-FLOAM

ABSTRACT

This thesis introduces the problem of simultaneous localization and mapping (SLAM) with focus on the use of a 3D LiDAR sensor. Firsly is written an introduction to SLAM itself and explained graph-based SLAM and dense map representation. The two most common point cloud alignment algorithms ICP and NDT are described. Then research of existing projects solving this problem is carried out. Described projects are all open-source and most of them support the ROS system. One of the described projects, Optimized SC-F-LOAM is explained in detail. Thesis describes it's odometry FLOAM and connection between it and graph optimization with loop closure. For loop closure is used descriptor ScanContext. Then it is presented design for implementing choosen project on offline and online datas from indoor. In last chapters is described proces of implementing and tuning project and at the end results of using said project in indoors are presented.

KEYWORDS

SLAM, 3D LiDAR, LiDAR odometry, point cloud, ROS, localization, mapping, Optimized SC-F-LOAM, F-LOAM, indoor SLAM, FLOAM, SC-FLOAM

MINAŘÍK, Jakub. *Sebelokalizace a mapování ve vnitřním prostředí na bázi LiDAR snímačů*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2023, 76 s. Diplomová práce. Vedoucí práce: Ing. Adam Ligocki, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Jakub Minařík
VUT ID autora:	203294
Typ práce:	Diplomová práce
Akademický rok:	2022/23
Téma závěrečné práce:	Sebelokalizace a mapování ve vnitřním prostředí na bázi LiDAR snímačů

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době se mobilní roboty stávají více nedílnou součástí světa kolem nás. Některé oblasti si bez nich nedokážeme představit a roboty do nich patří již desítky let, například vesmírný výzkum. Jejich nejčastější uplatnění je všude v oblastech lidem nepřístupných nebo nebezpečných. Využití mobilních robotů se nevyhýbá ani našim domácnostem a někteří z nás si již nedovedou život představit bez robotického vysavače či sekačky. Při pohledu do blízké budoucnosti se stále více začíná mluvit o zařazení autonomních vozidel do silničního provozu.

Mobilní roboty se obvykle navigují v prostředí za pomoci již známých map, GNSS snímačů nebo kontrolních bodů. Pokud se robot nachází v neznámém prostředí, k úloze navigace se přidává i úloha vytváření mapy, tomu se přezdívá sebelokalizace a mapování neboli SLAM. SLAM se velmi často používá pro mapování neznámých prostředí, například po zemětřesení, při mapování dna oceánu nebo třeba ve zmíněných vysavačích.

Tato práce je zaměřená na SLAM, který využívá jako snímač pro mapování a lokalizaci 3D LiDAR. Použití 3D LiDAR umožňuje robotu pohybovat se v prostředích nezávisle na světelných podmínkách, čehož může být využito při práci během večera a rána, kdy jsou světelné podmínky velmi proměnlivé. Použití 3D LiDARu umožňuje vytváření přesných 3D modelů prostředí. Největším problém s kterým se potýká SLAM s 3D LiDARem je výpočetní náročnost a zároveň dostatečná přesnost lokalizace a vytvořené mapy. Zlepšování výpočetní náročnosti problému umožňuje použití metod v reálném čase, ale i snížení velikosti a váhy robotu, což zpřístupňuje aplikaci ve více aplikacích jako jsou malé průzkumné roboty a drony.

Diplomová práce se bude věnovat existujícím open-source projektům řešícím SLAM s použitím 3D LiDARu. Nejprve bude vysvětlena teorie a princip za SLAMem a následně budou shrnuty existující projekty, kde budou popsány momentální přístupy a způsoby řešení této úlohy s 3D LiDARem. Z těchto projektů bude vybrán jeden, který bude podrobně prostudován a popsán. Probrána bude jeho momentální implementace včetně potřebných vstupů a výstupů. Následně bude cílem práce zprovoznit daný projekt. Ověření funkčnosti projektu započne již vytvořením návrhu hardwarového a softwarového řešení. Po provedení návrhu se bude projekt oživovat a testovat na několika dodaných záznamech. Po úspěšném testování na offline datech bude projekt přenesen na robota a vyzkoušen na datech v reálném čase. Na závěr práce pak budou zhodnoceny výsledky offline a online dat a rozdíly mezi nimi, budou diskutována možná použití projektu v jiných prostředích, jeho nedostatky a přednosti.

1 Metody

V této kapitole bude vysvětlena základní teorie týkající se SLAM systémů se zaměřením na technologie používané s 3D LiDARy. Poté budou popsány jednotlivé projekty.

1.1 SLAM

Sebelokalizace a mapování, jak již název napovídá, se skládá ze dvou částí. První část, lokalizace, odhaduje stav robota v mapě, druhá část vytváří zmíněnou mapu. Stav robota je nejčastěji popisován pozicí a orientací získanými z odometrie, ale může se skládat i z dalších parametrů jako je rychlost nebo kalibrační parametry.

Mapa může být vytvářena zaznamenáváním mezníků a značek, často určených pro orientaci robota. Mnohem častější, zvláště pak pro 3D LiDAR, je tvz. *dense SLAM*, který vytváří mapu z velkého množství bodů.

Obvyklou architekturu SLAM systémů lze rozdělit na dva prvky: *frontend* a *backend*. Obvyklým úkolem pro frontend je zpracování data ze snímačů, získání význačných bodů, asociace měření a specifického bodu odometrie a hledání smyček při návštěvě známých oblastí, tzv. *loop closure*. Backend obvykle stojí za pravděpodobnostní logikou algoritmu a vytvářením mapy.

Existuje několik možností jak řešit backend SLAMu. Zde bude popsán SLAM řešený pomocí grafu (*graph based SLAM*). Dalšími možnostmi, jak řešit SLAM, jsou například Kalmanův filtr nebo částicový filtr. Ty jsou pouze ojediněle používané s 3D LiDARem kvůli náročnému zpracování dat a kvůli malé přesnost s tímto snímačem.

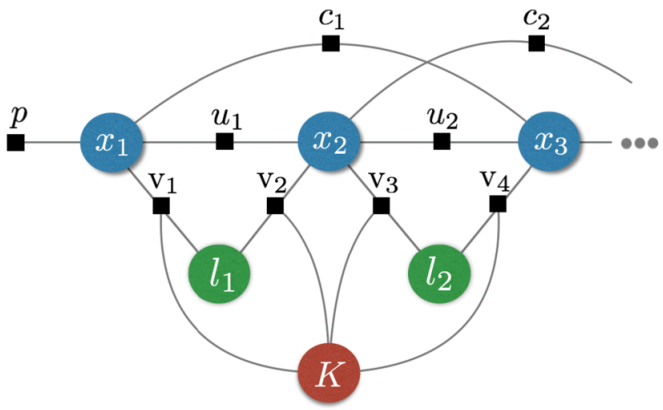
Dalším důležitým pojmem v oblasti SLAMu je tzv. *loop closure*. Jedná se o hledání a uzavírání smyček v mapě. Tedy pokud se robot vrátí na místo, kde již byl, a má ho zmapované, tak nalezení smyčky umožňuje opravu nahromaděné chyby mezi prvním a druhým průjezdem. [1]

1.1.1 Graph based SLAM

V dnešní době nejčastější přístup k problematice SLAMu je jako k problému maximálního posteriorního odhadu neboli *MAP*(*Maximum a Posteriori Estimation*) a velmi často je používán graf faktorů ke znázornění vzájemných vztahů mezi proměnnými.

Na obrázku 1.1 je schéma SLAMu jako grafu faktorů. Modré vrcholy znázorňují pozice robota za sebou v čase $(x_1, x_2, ...)$. Zelené vrcholy zobrazují orientační body $(l_1, l_2, ...)$ a červený vrchol jsou kalibrační parametry snímače (K). Hrany znázorňují

vztahy mezi jednotlivými vrcholy. Mezi pozicemi to jsou faktory odpovídající odometrii $(u_1, u_2, ...)$, mezi pozicemi a orientačními body pak faktory, které vyjadřují pozorování snímače $(v_1, v_2, ...)$. Hrany mezi vrcholy pozic popsané jako $(c_1, c_2, ...)$ označují nalezené smyčky. Pod p se pak skrývají předchozí faktory, například počáteční podmínky.



Obr. 1.1: SLAM jako faktorový graf [1]

Úkolem SLAMu je odhadnout neznámou proměnnou X, která obvykle vyjadřuje trajektorii jako diskrétní množinu pozic a pozici orientačních bodů. K dispozici máme množinu měření $Z = \{z_k : k = 1, ..., m\}$, kdy každé měření může být vyjádřeno jako funkce X, tedy jako

$$z_k = h_k(X_k) + \epsilon_k \tag{1.1}$$

kde $X_k \subseteq X$ je podmnožinou proměnných, h_k je známá funkce měření a ϵ_k je šum měření. Pomocí MAP odhadu se odhad X počítá jako proměnná $X^\star$, který je maximem posteriorní pravděpodobnosti $p(X|Z)$. V rovnici

$$X^\star \doteq \arg\max_X p(X|Z) = \arg\max_X p(Z|X)p(X) \tag{1.2}$$

je vyjádření MAP odhadu pomocí Bayesova teorému, kde $p(Z|X)$ je měření Z pro X a $p(X)$ je pravděpodobnost jevu X, která je konstantou pokud o jevu X nemáme žádné předchozí znalosti. Rovnice 1.2 se dá zjednodušit na rovnici

$$X^\star = \arg\min_X \sum_{k=0}^{m} ||z_k = h_k(X_k) + z_k||^2_\Omega \tag{1.3}$$

za předpokladu nezávislosti měření Z a pokud je šum měření ϵ_k Gaussův šum s nulovou střední hodnotou a s informační maticí Ω_k. Je použit zápis zjednodušení kde

$||e||^2_\Omega = e^T \Omega e$. Zde člen sumace vyjadřuje odhad pravděpodobnosti jednotlivých měření. Hledání minima této rovnice je obvykle řešeno postupnou linearizací například Gauss-Newtonovou nebo Levenberg-Marquardtovou metodou [1].

1.1.2 Dense SLAM

Mapa je reprezentována pomocí souřadnicové sítě vyjadřující obsazenost, mračna bodů, povrchové mapy nebo i nezpracovaná data ze snímačů. Výhodou této reprezentace mapy je lepší možnost dalších použití, například při plánování trasy nebo vizualizaci pro operátora. Robot se obvykle v této mapě lokalizuje pomocí hledání transformace mezi mapou vyjádřenou v mračnech bodů a mračnem získaným ze snímače. Tato transformace je obvykle hledána pomocí iterace pro nejpravděpodobnější rotaci a posunutí. Po odhadu pozice je mapa rozšířena o nová neznámá místa z mračna ze snímače.

Je několik možností jak získat transformaci mezi aktuální pozicí a mapou, jinak řečeno předešlou pozicí, která je již součástí mapy. Nejvíce naivním způsobem je porovnávat data ze skenu předešlé pozice s daty aktuální pozice, jehož nevýhodou může být znatelně menší přesnost. Další možností je porovnávat momentální sken k celé globální mapě. To ale může být výpočetně velmi náročné. Nejčastěji se proto volí vytváření lokální mapy z několika předešlých pozic, k níž se pak porovnávají data z aktuální pozice. Podobně se pak hledá i transformace mezi aktuální pozicí a pozicí, se kterou byla uzavřena smyčka na mapě. Jedna z nejčastějších metod, jak hledat transformaci mezi dvěma mračny bodů, je Iterative closest point (ICP) nebo normal distribution transform (NDT).

1.1.3 Iterative closest point

ICP se používá pro minimalizaci rozdílu mezi dvěma mračny bodů, případně i mezi mračny bodů s povrchem. Existuje mnoho variant algoritmu pro různé situace, s různými řešeními jednotlivých kroků iterace, proto bude popsána pouze základní kostra algoritmu. Pro funkci algoritmu je většinou potřeba mít počáteční odhad transformace mezi mračny. Nejprve je vybráno jedno mračno, ke kterému se hledá transformace. Druhé mračno je každou iteraci rotováno a posunuto nalezenou transformací. Jednotlivé kroky algoritmu jsou [2]:

1. Výběr několika bodů v jednom nebo v obou mračnech.
2. Nalezení vhodných protějšků vybraných bodů v druhém mračnu.
3. Přiřazení vhodných vah jednotlivým párům a vyřazení nevhodných párů.
4. Vypočítání chyby vhodnou metrikou, například vypočítání euklidovské vzdálenosti mezi body.

5. Minimalizace chyby, tedy nalezení transformace, která má lepší chybu, než byla vypočtena momentální.
6. Algoritmus iteruje znovu od 1.bodu, dokud není splněna podmínka pro ukončení.

1.1.4 Normal distribution transform

Prvotní verze algoritmu byla představena pro 2D registraci bodů, ale později byla rozšířena na 3D. Stejně jako u ICP existuje mnoho variant a modifikací. Zde bude popsán základ algoritmu pro 2D prostor, pro 3D je postup podobný.

Mapování do bodů v prostoru do NDT jej prvně rozdělí na pravidelné buňky. Pro každou buňku je vypočítána střední hodnota q a kovarianční matice Σ z bodů, které obsahuje. Pravděpodobnost, že se bude změřený bod x nacházet v dané buňce, je modelována normální rozložením $N(z, \Sigma)$:

$$p(x) \sim \exp\left(-\frac{(x-q)^T\Sigma^{-1}(x-q)}{2}\right) \tag{1.4}$$

Takto je získána po částech souvislá diferencovatelná funkce pravděpodobnosti daného prostoru. Pro zmenšení diskretizace se jednotlivé buňky volí tak, aby se překrývaly, což je pro účely vysvětlení principu zanedbáno. Algoritmus se dá rozdělit do několika kroků:

1. Vytvoření NDT z prvního skenu.
2. Inicializace prvotního odhadu pro otočení a posunutí.
3. Mapování bodů druhého skenu do souřadnic prvního pomocí transformace otočení a posunutí na základě prvotního odhadu.
4. Nalezení odpovídajícího normálního rozložení pro každý z mapovaných bodů v NDT.
5. Vypočítá se skóre, které je součet pravděpodobností z nalezených normálních rozložení.
6. Nalezení lepšího skóre, tedy výpočet nového odhadu za účelem lepšího skóre.
7. Opakování od 3. bodu, dokud nejsou splněny podmínky pro ukončení.

1.2 Popis projektů

V této části budou popsány open-source projekty, které řeší SLAM s použitím 3D LiDAR snímače, se zaměřením na ty, které je možné implementovat do systému ROS.

1.2.1 LOAM

Nejedná se o úplný SLAM, jelikož metoda neřeší uzavírání smyček. Pro zmenšení výpočetní náročnosti a zlepšení výkonu metody je algoritmus rozdělen na dvě části, běžící paralelně. Jedna určuje odometrická data s horší přesností při frekvenci 10 Hz. Druhá část daná data zpřesňuje a vytváří mapu. Oproti první části beží na pomalejší frekvenci 1 Hz. Pro další odhad v první části jsou pak již použita data zpřesněná druhou částí. Výpočet odometrie je prováděn na základě hledání význačných bodů a jejich následným párováním mezi skeny [3].

Z této metody vychází velké množství dalších metod a je implementováno více variant. Jedna z variant je specializovaná na Velodyne snímač [4].

A-LOAM

Varianta A-LOAM používá Eigen a Ceres Solver knihovny pro zjednodušení zdrojového kódu a jeho optimalizaci. Cílem této varianty není upravit fungování algoritmu, ale zlepšit jeho naprogramování, v rámci optimalizace i přehlednosti kódu [5]. Často jsou tyto dvě verze algoritmu zaměňovány, protože z matematického hlediska mezi nimi není rozdíl a algoritmus nepřináší nové poznatky. Pro A-LOAM existuje verze s implementací detekce smyček pomocí deskriptoru ScanContext a s ním souvisejícím vyhledávacím algoritmem. Jeho cílem je hledat, zdali nebyla lokace již navštívena [6].

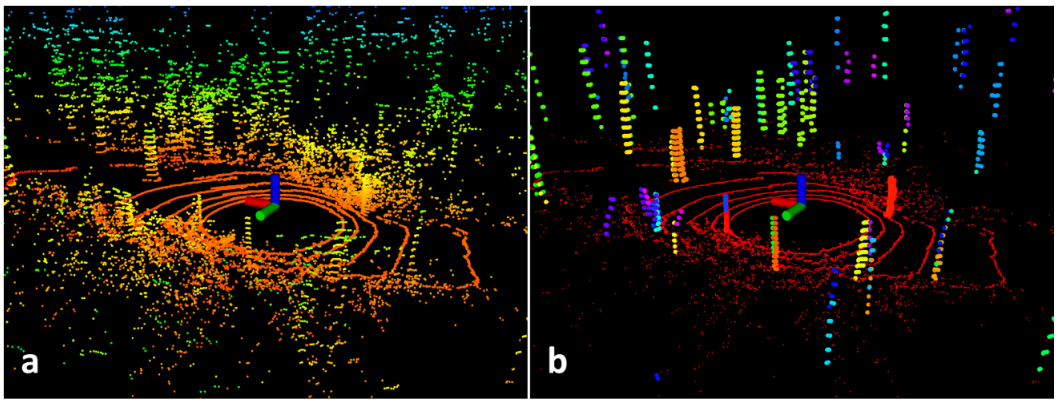
1.2.2 LeGO-LOAM

Tato metoda vychází z metody LOAM. Specializuje se na malé neřízené pozemní roboty ve složitém prostředí, například ve venkovní prostředí, a zaměřuje se na menší výpočetní náročnost. Nejprve je nalezena země a na ostatní data je aplikována segmentace, podobná jako při zpracování obrazu, pro odstranění šumu. Z roviny země je pak určena transformace souřadnic náklonu a z ostatních dat zbylé transformace ostatních souřadnic. Pro zmenšení posunu od správné polohy za čas se zavádí hledání smyček, které je implementováno naivní verzí algoritmu ICP [7].

Metoda je implementována v systému ROS. Stejně jako u předešlé metody existuje verze, která je optimalizovaná a v rámci kódu zpřehledněná [8]. Pro LeGo-LOAM existuje také implementace detekce smyček pomocí ScanContext [9].

1.2.3 F-LOAM

F-LOAM je další krok v optimalizaci, který vychází z původního LOAM a z A-LOAM. Při odhadu transformace mezi skeny, pro zmenšení zkreslení skenu, je použita dvoukroková kompenzace zkreslení. Z předchozích dvou transformací je vypočten koeficient, který je pak použit při lineární interpolaci odhadu transformace.



Obr. 1.2: LeGO-LOAM - Původní mračno bodů na části *a*, v části *b* jsou červeně označeny body, které představují zemi, ostatní body jsou ty, které zůstaly po segmentaci [7]

Daný způsob je výpočetně méně náročný a podobně přesný jako obvykle používaný iterační odhad. Odhad pozice je vypočten z význačných hran a ploch, které jsou porovnávány k lokálním mapám význačných hran a bodů. Navíc je zavedeno váhování význačných bodů a hran. Algoritmus je implementován v systému ROS [10].

1.2.4 Optimized SC-F-LOAM

Metoda kombinuje lokalizaci z F-LOAM, pomocí deskriptoru ScanContexu implementuje hledání smyček a pro výpočet backendu na základě faktorového grafu používá knihovnu GTSAM. Při hledání transformace pozice pro nalezenou smyčku je použito porovnání podle významných bodů. Zavádí adaptivní vzdálenostní práh pro hledání smyček. Algoritmus pracuje v systému ROS [11].

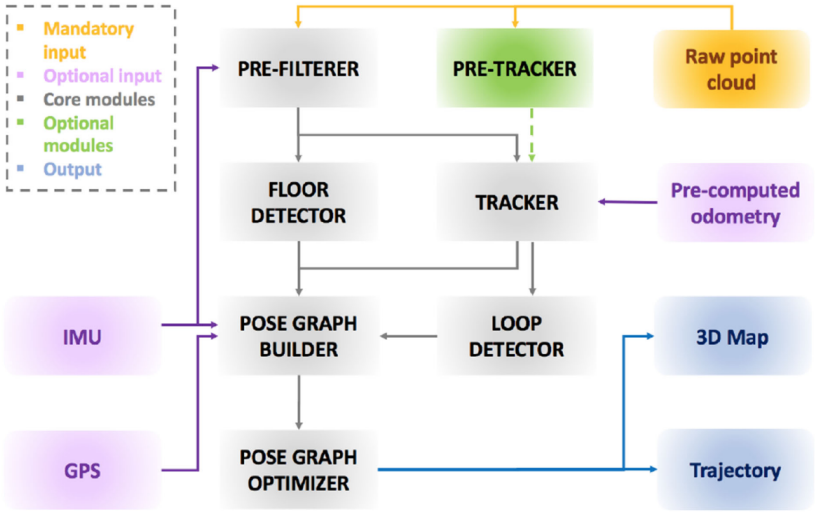
1.2.5 HDL graph SLAM

Metoda implementuje SLAM na základě grafu. Získané mračno bodů je prvně filtrováno a následně porovnáno s předchozím mračnem. Pro získání transformace mezi dvěma mračny je použita transformace normálních rozložení (*Normal distributions transform - NDT*). Vrcholy grafu jsou pozice zjištěné z transformace. Kandidáti pro případné smyčky jsou vybíráni podle translační vzdálenosti a délky dráhy mezi vrcholy grafu. Graf je aktualizován vždy když je nalezena smyčka.

Pro vnitřní prostory je možné použít omezení, které počítá s tím, že se země nachází v jedné rovině, tato rovina je nalezena pomocí RANSAC. Metoda umožňuje integraci IMU dat a GPS snímače [12]. HDL graph SLAM je plně integrován s ROS systémem [13].

1.2.6 ART-SLAM

Metoda vychází z HDL graph SLAMu, pro zjištění odometrie hledá orientaci a posunutí mračen bodů. ART-SLAM umožňuje velkou modularitu, protože se skládá z několika na sobě minimálně závislých bloků. To umožňuje změnu bloku, případně jeho úpravu při dodržení správného formátu toku dat mezi jednotlivými bloky.



Obr. 1.3: ART-SLAM - Schéma modulů [14]

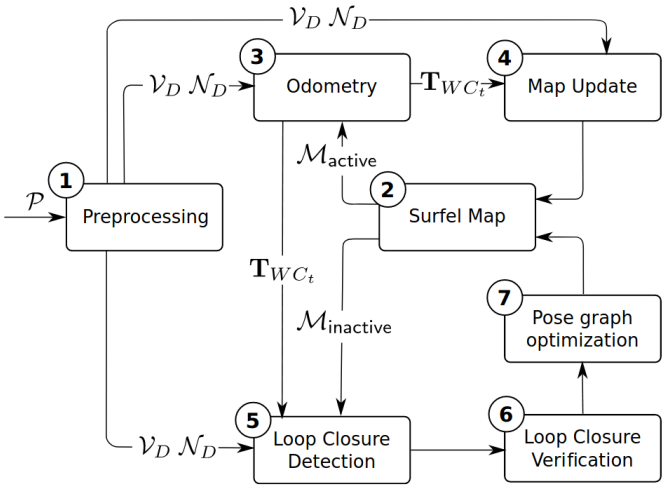
Na obrázku 1.3 jsou vidět jednotlivé moduly metody. Šedé moduly jsou nezbytné pro metodu a tvoří její jádro. Zeleně je označen volitelný modul *pre-tracker*, který získává hrubý odhad pohybu. Oranžově označený je povinný vstup zastupující mračno bodů. Volitelné vstupy mají fialovou barvu, tedy IMU, GPS a počáteční odhad odometrie (*pre-computed odometry*). Výstupy jsou zaznačeny modrou barvou a je to 3D mapa a trajektorie pohybu. Modul *pre-filterer* filtruje data pro zmenšení velikosti a snížení šumu mračna bodů. Modul *floor detector* detekuje zemi a souřadnice na základě polohy vůči ní. *Tracker* odhaduje polohu na základě mračna bodů. *Pose graph builder* vytváří graf z odometrie. *Pose graph optimizer* optimalizuje vrcholy grafu zastupující pozici robota.

ART-SLAM je nezávislý na systému ROS, ale je pro něj poskytnuta nástavba pro použití se systémem ROS [14].

1.2.7 SuMa

Metoda zaměřující se na městské prostředí vytváří mapu ze surfelů. To metodě umožňuje efektivnější reprezentaci velkých prostředí. Metoda vytváří graf pozic, který optimalizuje a aktualizuje. SuMa neintegruje ROS systém.

Na obrázku 1.4 se nachází schéma architektury algoritmu. V části *Preprocessing* je z mračna bodů $\mathcal{P}$ je vygenerována vortexová $\mathcal{V}_D$ a normálová $\mathcal{N}_D$ mapa. Odometrie je vypočítana pomocí ICP z normálové a vortexové mapy skenu a z normálové a vortexové mapy získané z aktivní části celkové surfelové mapy $\mathcal{M}_{active}$, což je část mapy, která představuje několik posledních mračen bodů. V části *Map Update* je



Obr. 1.4: SuMa - Schéma [15]

aktualizována surfelová mapa ze zjištěné odometrie a vortexové a normálové mapy. Detekce smyček je provedena z neaktivní části mapy $\mathcal{M}_{inactive}$ a následně je ověřena ve skenech v modulu *Loop Closure Verification* [15].

1.2.8 SuMa++

Jedná se o rozšíření předchozí metody o sémantické informace.

Výsledkem je mapa s označenými názvy povrchů, což umožňuje lepší odfiltrování pohybujících se objektů a menší chybovost při porovnávání skenů na základě přidané podmínky sémantiky. Následkem toho dochází ke zlepšení nejen v rámci vyhledávání smyček ale i v rámci odometrie. Metoda spojuje data předchozího projektu SuMa se sémantickou informaci generovanou z plně konvoluční neuronové sítě (*FCN*) [16].

1.2.9 LOAM Livox

Na rozdíl od předchozích se metoda zaměřuje na solid-state LiDARy, obecněji na LiDARy s malým zorným polem. Rozdíl mezi obvyklými LiDARy a solid-state LiDARy bude popsán v následující kapitole. Po výběru bodů, které nejsou na hraničních podmínkách, například body blízko okraje zorného pole, jsou nalezeny význačné hrany a plochy. Pro zmenšení nevýhody zorného pole, které zabírá méně význačných bodů,

je použita reflektivita, případně intenzita, vracená snímačem jako čtvrtý parametr pro hledání význačných bodů. To znamená, že hrana může být nalezena při přechodu materiálů. Jednotlivé význačné hrany a plochy jsou ukládány v KD stromech pro efektivní vyhledávání při porovnávání. Metoda zavádí i možnosti kompenzace pro rozostření pohybem. Dalším rozdílem oproti standardnímu LOAM algoritmu je zavedení paralelizace. Metoda je vytvořena v systému ROS [17].

1.2.10 SSL SLAM

Další metoda, která se zaměřuje na solid-state LiDARy. Stejně jako u metody LOAM i zde je odometrie vyhledává na základě význačných hran a ploch. Ty jsou porovnávány s lokální mapou sestavenou z několik blízkých snímků. Globální mapa je aktualizována pouze klíčovými snímky. Ty jsou určeny buď značnou změnou v pozici, značnou změnou v rotaci, nebo po určité časové době. V praxi se první dva parametry se definují podle zorného pole snímače a časová doba dle výpočetních možností. Metoda je vyvinuta v rámci systému ROS [18].

1.3 Výběr Metody

Při výběru metody byly hned na začátku vyřazeny SuMa a SuMa++, protože nepodporují systém ROS. Další problémem u SuMa++ je model neuronové sítě, který, ač je poskytnut natrénován, je vytvořen pro městské prostředí. Pro jiné prostředí by bylo třeba natrénovat nový model.

Z předchozích metod nejlépe v rámci přesnosti vychází metody Optimized Sc-F-LOAM a ART-SLAM, které byly při publikování testovány na KITTI datasetech. Výsledky na těchto datasetech byly porovnány s výsledky ostatních projektů [11] [14]. Pro odlišné vyhodnocení chyb je ale těžké mezi sebou metody porovnat.

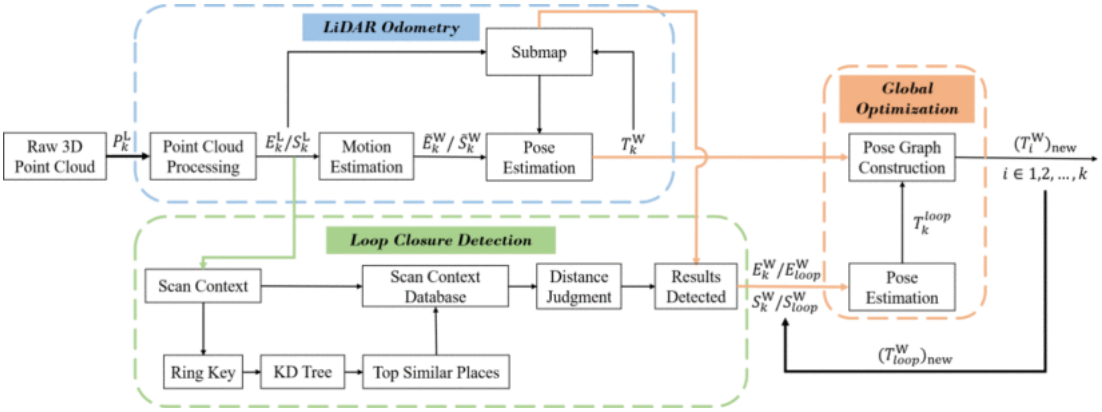
Z pohledu výpočetní náročnosti je na tom pravděpodobně nejlépe LeGO-LOAM, který byl vytvořen pro vestavěné systémy v malých robotech. V experimentech tedy vychází jako nejméně výpočetně náročný, ale tento parametr nepatří vždy mezi ty které jsou vyhodnocovány, proto jej nelze s jistotou potvrdit. Například oproti F-LOAM, HDL graph SLAM a A-LOAM je LeGO-LOAM výpočetně méně náročný [10].

Proti ART-SLAM pak hovoří to, že pro ROS systém je poskytnuto pouze základní přemostění, které neslibuje veškerou funkcionalitu, ale pouze základní. Nicméně to nevylučuje riziko, že v budoucnu může být nutné dodělat podporu některých funkcí, nebo případně brát v potaz ústupky. Z tohoto důvodu se jeví metody plně integrované v ROSu jako lepší.

Proto nakonec byla vybrána metoda Optimized Sc-F-LOAM, který slibuje největší přesnost a plnou integraci v systému ROS. Nevýhodou může být, že patří k nejpozději zveřejněným projektům ze všech zmíněných a může se objevit více problémů než u metody, která již byla mnohokrát použita. Ale je třeba brát ohled k tomu, že metoda vychází již z existujících projektů a mění jen některé části algoritmu.

2 Optimized SC-F-LOAM

Projekt je implementována a odladěna pro použití na KITTI datasetech [19], tedy na venkovní prostory, pro snímač na střeše pomalu jedoucího auta. Nicméně v publikaci je psáno, že je možné použití ve vnitřních prostorách. V kódu jako takovém jsou k některým veličinám připojeny komentáře s přibližnými hodnotami pro použití ve vnitřních prostorách. Metoda by se dala rozdělit do tří částí na odometrii FLOAM, hledání smyček, které je postaveno na deskriptoru ScanContext a na vytváření a optimalizaci grafu pozic za pomoci GTSAM. Tyto tři části běží paralelně. Podrobné schéma celého algoritmu je znázorněné na obrázku 2.1.



Obr. 2.1: Optimized SC-F-LOAM - Schéma - Modrá část značí odometrii, zelená hledání smyček a oranžová optimalizaci a vytváření grafu pozic [11]

2.1 Odometrie

Odometrie je stejná jako u metody FLOAM. Skladá se ze čtyř částí.

2.1.1 Zpracování mračna bodů

Mračno bodů je zpracováno (*Point Cloud Processing*) nalezením významných bodů a odfiltrováním nevýznamných. Významné body jsou získány na základě rozdílu funkce hladkosti povrchu. Vzorec pro výpočet hladkosti je

$$\sigma_k^{(m,n)} = \frac{1}{\left|S_k^{(m,n)}\right|} \sum_{p_k^{(m,j)} \in S_k^{(m,n)}} \left(\left\| p_k^{(m,j)} - p_k^{(m,n)} \right\| \right), \tag{2.1}$$

kde $S_k^{(m,n)}$ jsou okolní body bodu $p_k^{(m,n)}$ ve vodorovném směru, $\left|S_k^{(m,n)}\right|$ je počet bodů v okolí. Protože LiDAR má mnohem větší hustotu bodů ve vodorovném směru je hladkost lokálního povrchu počítána pouze na vodorovné rovině.

Hodnota hladkosti pak určuje rozdělení bodů, pokud body přesahují určité meze, jsou brány jako body hran (E_k^L), nebo naopak jako body ploch (S_k^L). Ostatní body jsou vyřazeny jako zašuměné nebo nevýznamné [11].

2.1.2 Odhad pohybu

Vstupem části odhad pohybu (*Motion Estimation*) jsou odfiltrovaná a rozdělená mračna bodů na body hran a ploch (E_k^L, S_k^L). Úkolem je transformovat mračna hran a ploch a odstranit jejich zkreslení způsobené pohybem snímače. Je předpokládáno, že většina 3D LiDARů je schopna pracovat při frekvenci $10Hz$ a více. Proto je pro výpočet tohoto odhadu bráno, že úhlová a lineární rychlost jsou během této krátké doby konstantní. Na základě toho odhadu je částečně kompenzováno zkreslení mračen. Zároveň jsou mračna hran a ploch ($\tilde{E}_k^L$, $\tilde{S}_k^L$) transformovány do globálních souřadnic [11].

2.1.3 Odhad pozice

Vstupem části odhadu pozice (*Pose Estimation*) jsou transformovaná mračna hran a ploch ($\tilde{E}_k^W$, $\tilde{S}_k^W$) a mapa rozdělená na hrany a plochy z modulu *Submap*. Pro každý bod z mračen $\tilde{E}_k^W$, $\tilde{S}_k^W$ je vytvořena kovarianční matice jejich blízkých bodů z mapy. Pokud jsou body v řadě, je jedno vlastní číslo kovarianční matice mnohem větší. Vlastní vektor n_E^g spojený s tímto číslem určuje orientaci dané hrany. Pozice hrany p_E^g je zvolena jako geometrický střed vybraných blízkých bodů. Pro plochy se hledá nejmenší vlastní číslo kovarianční matice. Takto získané optimální hrany a plochy umožní získání spojení mezi globálními body mapy a mračny $\tilde{E}_k^W$, $\tilde{S}_k^W$. Minimalizací vzdálenosti mezi těmito body je získán odhad transformace mezi mapou a pozicí, při které byly pořízeny mračna $\tilde{E}_k^W$, $\tilde{S}_k^W$. Vzdálenost mezi bodem p_E z mračna $\tilde{E}_k^W$ a optimální hranou je

$$f_E(p_E) = p_n \cdot ((T_k^W p_E - p_E^g) \times n_E^g), \tag{2.2}$$

kdy T_k^W je matice transformace mezi optimální hranou a vstupním bodem hrany, p_E je vstupní bod hrany, p_n je vypočítán jako normalizace vektoru

$$p_n = \frac{(T_k p_E - p_E^g) \times n_E^g}{||(T_k p_E - p_E^g) \times n_E^g||}. \tag{2.3}$$

Vzdálenost bodu p_S mračna $\tilde{S}_k^W$ a optimální plochy je vypočítána

$$f_S(p_S) = ((T_k^W p_S - p_S^g) \cdot n_S^g), \tag{2.4}$$

kdy p_S^g je pozice optimální plochy a n_S^g její normála. Pro zlepšení odhadu jsou body váženy již dříve zmíněnou a počítanou hladkostí 2.1, protože body s lepší hladkostí

jsou častěji registrovány v po sobě jdoucích skenech. Váha pro hrany je počítána rovnicí

$$W(p_E) = \frac{exp(-\sigma_E)}{\sum_{p^{(i,j)} \in E_k} exp(-\sigma_k^{(i,j)})}, \tag{2.5}$$

rovnice pro výpočet váhy pro plochy se liší pouze opačným znaménkem v obou exponenciálních funkcích. Výsledná rovnice pro minimalizaci je

$$\min_{T_k^W} \sum W(p_E) f_E(p_E) + \sum W(p_S) f_S(p_S), \tag{2.6}$$

ta je pak řešena pomocí Gaussovy-Newtonovy metody.[10]

2.1.4 Aktualizace submapy

Tato část má na vstupu mračna významných bodů skenu (E_k^L, S_k^L) a transformaci T_k^W mezi souřadnicovým systémem skenu z LiDARu a souřadnicemi submapy. Mračna významných bodů jsou transformována a přidána do submapy.[11]

2.2 Hledání smyček

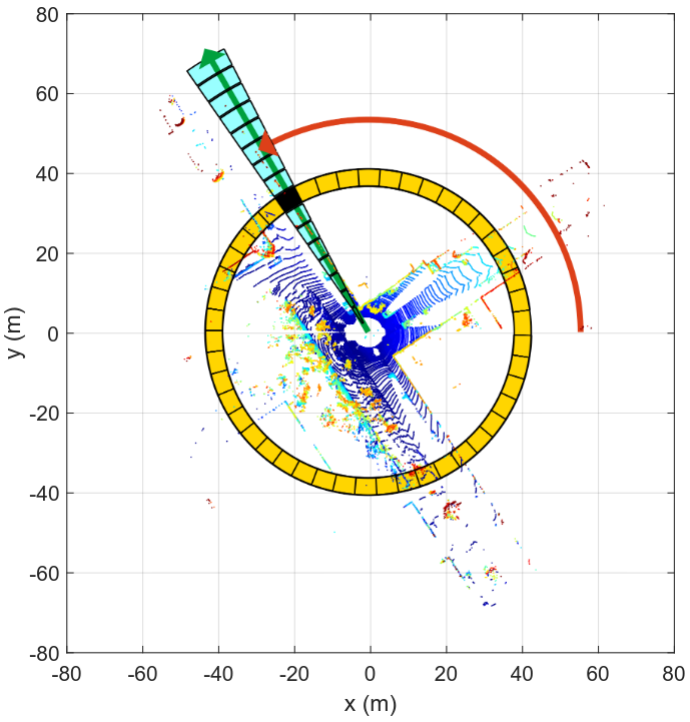
Detekce smyček je implementovaná a postavená na deskriptoru ScanContext.

2.2.1 ScanContext

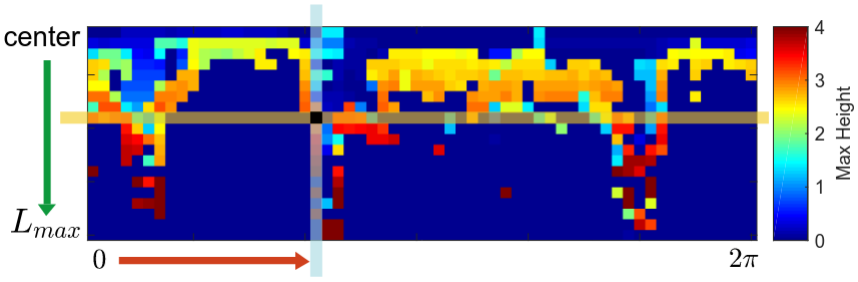
Proces vytváření deskriptoru ScanContext z mračna bodů se skládá ze dvou kroků. První krok, znázorněný na obrázku 2.2, rozdělí oblast skenu do bloků podle vzdálenosti a úhlu od snímače. Bloky jsou vytvořeny z pohledu seshora, čímž vznikne 2D prostor, a hodnotou bodů se stane jejich výška. Dále je prostor rozdělen na pravidelné výseče a mezikruží, jejich průnikem vznikají dané bloky. V druhém kroku, obrázek 2.3, se bloky poskládají do matice. V jednotlivých blocích jsou výšky bodů zprůměrovány a bloky jsou tak zastoupeny jednou hodnotou. Vybere se počáteční výseč s největší výškou. Z bloků je sestavena matice, body nejblíže středu jsou na prvním řádku a body v nejdál od středu na posledním. Sloupce určují úhel od počáteční výseče. Pro jednoduší představu je v obrázcích 2.2 a 2.3 černě zvýrazněn stejný blok, jeho výseč tyrkysově a mezikruží žlutě [20]. Oproti původnímu deskriptoru jsou v této metodě použita, pro vytvoření deskriptoru, pouze mračna s body hran a ploch. Hranice maximální vzdálenosti se určuje dynamicky [11].

2.2.2 Vyhledávání smyček

Kvůli rychlému vyhledávání je zavedeno ještě další zakódování deskriptoru ScanContext do tvz. *Ring key*, řádek z matice v obrázku 2.3, je zakódován do jedné hodnoty,



Obr. 2.2: Vytváření deskriptoru ScanContext - první krok [20]



Obr. 2.3: Vytváření deskriptoru ScanContext - druhý krok [20]

a vzniklý vektor, pro danou polohu je pak uložen do KD stromu. Tento postup je pak použit pro nalezení kandidátů pro smyčku. Jsou vyhledávány pouze stejné vektory. Kandidáti jsou pak porovnáni na základě vzájemných vzdáleností mezi jejich deskriptory ScanContext [20].

Na obrázku 2.1 v zelené části se pod blokem *Scan Context* skrývá vytvoření deskriptoru. Deskriptor je následně zařazen do databáze deskriptorů (*Scan Context Database*) a zároveň je z něj vypočítán *Ring Key*, který je uložen v KD stromu *KD tree*. Poté jsou porovnány jednotlivé *Ring Key* skenů a nalezeny nejvhodnější

kandidáti (*Top Similar Places*). Kandidáti jsou hodnoceni podle jejich vzájemné vzdálenosti (*Distance Jugment*). Pozice s nejlepším výsledekem je nalezenou smyčkou (*Result Detected*).

2.3 Vytváření a optimalizace grafu pozic

Graf pozic se skládá ze dvou částí konstrukce a optimalizace grafu (*Pose Graph Construction*) a odhadu pozice z nalezené smyčky (*Pose Estimation*).

2.3.1 Odhad pozice z nalezené smyčky

Hledá transformaci mezi aktuální pozicí mračen (E_k^W, S_k^W) a mračen z nalezené smyčky (E_{loop}^W,S_{loop}^W). pro odhad transformace se obě mračna musí nacházet ve stejném souřadném sytému, proto jsou mračna aktuální pozice transformována na globální systém podle výstupu z grafu pozic ($(T_loop^W)_new$). Je vytvořen úsek mapy obsahující mračna z aktuální pozice a ze smyčky. Pro výpočet odhadu transformace je pak použita stejná metoda jako při odometrii.[11]

2.3.2 Graf pozic

Vytváření grafu pozic je možné rozdělit na konstrukci a optimalizaci. Výsledkem jsou pak optimální transformace mezi pozicemi ($(T_i^W)_new$). Při konstrukci grafu jsou jednotlivé vrcholy pozicemi. Transformace mezi pozicí a smyčkami jsou hrany mezi vrcholy. Optimalizace grafu je prováděna pomocí knihovny GTSAM, která je vytvořená pro výpočty pro spojení oblastí počítačového vidění a robotiky, včetně SLAMu [11].

2.4 Implementace v systému ROS

V této části bude rozebrána implementace metody v systému ROS. Popsány budou jednotlivé části a jakou část metody provádí, vstupy a jejich formát, které jsou potřeba pro správnou funkčnost a jaké výstupy metoda poskytuje.

2.4.1 ROS

Před vysvětlením implementace metody zde bude popsána základní teorie a fungování systému ROS. Kapitola se bude věnovat především jeho částem důležitým pro pochopení implementace.

ROS, neboli Robot Operating System je operační systém, který poskytuje hardwarovou abstrakci, umožňuje nízkoúrovňové programování a posílání zpráv mezi

procesy. Momentálně existují dvě verze. První starší je označováná jako ROS 1 a je postupně nahrazována novější verzí ROS 2. Jedním z rozdílů je absence *ROS Master*, který plní úlohu hlavního řídícího procesu a správy komunikace v ROS 1. Mezi verzemi ROS 1 a ROS 2 existují tvz. mosty (*bridges*), které umožňují propojení projektů z jednoho systému s druhým. Vybraná metoda je vytvořena ve verzi ROS 1, dále tedy budou popsány základy verze ROS 1. ROS jako takový obsahuje i balíčky rozšiřující podporu o často používané funkce.

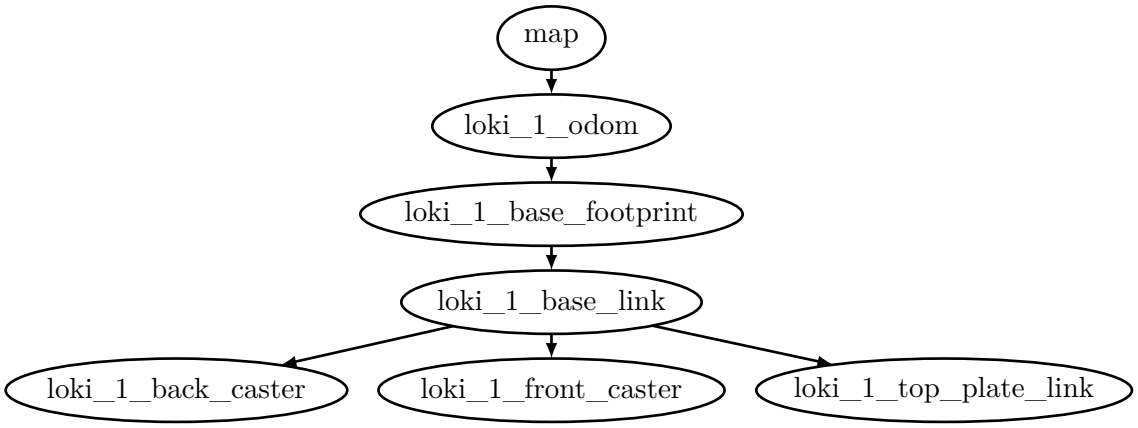
Jednotlivé procesy jsou nazývány *node*, neboli uzly. Komunikace mezi nimi bývá obvykle dvojího typu, a to pomocí *topic* a *service*. Topic používá model *publish/-subscribe*, kdy jeden node publikuje data pod daným názvem (*topic*) a pak ostatní node mohou daný *topic* odebírat, a tím získat data, která jsou publikována. Service, neboli služby poskytují typ komunikace dotaz a odpověď, node se dotáže na data a následně mu dotázaná node pošle odpověď s daty.

Při spouštění je v ROS1 vždy potřeba mít běžící ROS Master, který zajišťuje potřebnou komunikaci a běh celého systému. Spuštění ROSu často vyžaduje spouštění více procesů a nastavování složitější konfigurace, a proto je používán launch soubor, který tyto věci zabaluje, díky čemuž je možné spustit projekt jen jedním příkazem. Launch soubor je ve formátu xml a pomocí několika tagů umožňuje spuštění více uzlů zároveň, definování jejich parametrů, vytvoření namespace nebo spouštění na různých počítačích.

Výhodou ROSu je možnost nahrát publikované zprávy v závislosti na čase a vytvořit tak *bag file*, který umožňuje přenesení a pozdější puštění. To umožňuje například nahrát data ze snímačů a později pak ladit jejich zpracování, bez potřeby snímačů a stejných podmínek.

Transformace v ROSu

Pro jednoduší manipulaci s transformacemi mezi pozicemi se v ROSu používá balíček *tf*. Transformace v celém systému fungují jako jeden *topic*. Uvnitř něj jsou schované transformace k jednotlivým pozicím (*frame*). Transformace je vždy definována translací a rotací pro daný čas mezi dvěma danými pozicemi. Pozice jsou uskupeny do stromu, proto se při definici transformace udává, která pozice dědí z které. Ukázka takového stromu je na obrázku 2.4. Stromová struktura pro správnou funkčnost vyžaduje, aby všechny pozice měly společnou pozici, nacházející se na počátku stromu, nejčastěji počátek souřadného systému světa. Tím je zaručeno, že je možné dopočítat transformaci z jakékoliv pozice na jinou.



Obr. 2.4: Strom *tf* pro robota *loki* vztažený k pozici *map*

2.4.2 Optimized SC-F-LOAM v ROSu

Algoritmus je rozdělen na několik node, jak je znázorněno na obrázku A.1 v příloze. Obdélník znázorňuje odebírané zprávy a node je znázorněn elipsou. V grafu není zobrazen topic tf pro celkové zjednodušení, a také tam nejsou zobrazeny node, které vytváří pomocnou transformaci tf a vytváří trajektorii dodané transformace správné cesty a cesty odhadnuté pouze pomocí odometrie.

Vstupy a výstupy

Vstupem algoritmu je zpráva s topic *velodyne_points*, která je ve formátu *PointCloud2* definovaném v knihovně ROSu. Jedná se o mračno bodů podobně seřazených jako 2D obraz [21].

Algoritmus také pracuje s daty správné pozice, v kódu jako *ground truth*, která je reprezentována tf pozicí pod názvem *velodyne*. Z těchto dat pouze tvoří trajektorii pro porovnání a uložení ve stejném formátu jako výslednou trajektorii.

Výstupní zprávy jsou publikovány z node *sc_floam_laserLO*:

- *aft_pgo_map* - celá mapa po optimalizaci grafem, vzhledem k množství dat průběžně aktualizována a obnovována s frekvencí *0.1 Hz*
- *aft_pgo_path* - výsledná trajektorie robota, obnovována s frekvencí *10 Hz*
- *aft_pgo_odom* - momentální odometrie na základě optimalizovaných pozic, obnovována s frekvencí *10 Hz*

První dvě publikované zprávy jsou ve formátu *PointCloud2*. Poslední je ve formátu *Odometry* z knihovny ROSu *nav_msgs*.

Dalším možným výstupem je topic z node *sc_floam_laser_mapping_node*, a to s názvem */map*, ten je výsledkem algoritmu FLOAM, tedy pouze odometrie, bez hledání smyček a vytváření a optimalizace grafu. Algoritmus během své činnosti také

průběžně ukládá výsledná data na disk. Je ukládáno a průběžně přepisováno výsledné mračno bodů mapy ve formátu *pdb*. Dále jsou ukládány jednotlivé transformace pozic odometrie, pozic po optimalizaci a pravdivých pozic na základě zmíněného vstupu pravdy. Tyto transformace jsou ukládány do textového souboru, jednotlivé hodnoty jsou odděleny mezerou a každá nová pozice jen na novém řádku.

Struktura projektu

Na obrázku A.1 jsou zobrazeny jednotlivé node a jejich propojení. Jednotlivé node, a kterou část metody provádí, budou popsány v následující seznamu. Budou popsány i node, které vychází z balíčku ROSu a na obrázku nejsou zobrazeny.

- *sc_floam_laser_processing_node* filtruje vstupní mračno bodů a rozděluje jej na hrany a body.
- *sc_floam_odom_estimation_node* počítá odhad odometrie z mračen hran a bodů.
- *sc_floam_laser_mapping_node* vytváří mapu z odometrie.
- *sc_floam_laserLO* hledá smyčky a provádí konstrukci a optimalizaci grafu.
- *word2map_tf* vytváří transformaci mezi pozicí světa a vytvářenou mapou, využívá příkaz *static_transform_publisher* pro vytvoření node, která publikuje statickou transformaci.
- *gt/trajectory_server_loam* zaznamenává trajektorii transformace pozice dle dodané správné pozice, využívá *hector_trajectory_server*, který zaznamenává polohu frame z tf.
- *base_link/trajectory_server_loam* zaznamenává trajektorii transformace pozice dle odometrie, bez optimalizace pomocí hledání smyček a grafu, taktéž je využit *hector_trajectory_server*.

Poslední tři node by z podstaty metody neměly být nutné pro fungování algoritmu, ale zjednodušují práci s ním a vyhodnocování výsledků.

3 Návrh

V této kapitole bude popsán dostupný hardware a jeho výběr. Následně bude navržena software architektura výsledného systému.

3.1 Výběr hardware

Vybraná metoda, byla při publikaci testována na notebooku s 12-jádrovým procesorem AMD R5-5600H a s 16 GB paměti RAM [11]. Pro účely této práce bude použit notebook s 8-jádrovým procesorem Intel i7-10510U a s 16 GB paměti RAM na offline datech i na online datech. Testování na online datech bude provedeno za pomoci mobilního robota Loki. Robot váží zhruba 10 kg a má diferenciální podvozek.

3.1.1 3D LiDAR

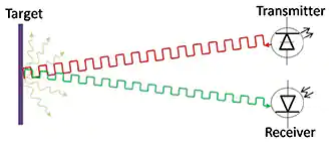
Jednou z populárních variant, jak vytvořit model prostředí, je použití 3D LiDARu. Technologie LiDAR funguje na principu *time of flight* (ToF). ToF snímač měří pomocí světla vzdálenost bodu. V LiDARu je použit takový snímač jednou nebo vícekrát, a s jeho pomocí je měřena velká část okolních bodů. Existuje i varianta 2D, ve které jsou měřeny vzdálenosti bodů v jednom řádku, obvykle ve stejné výšce. Ve 3D LiDARech je přidána souřadnice výšky, často přidáním řádků, v kterých je měření prováděno. Existují dva typy 3D LiDARů elektromechanické a solid-state.

Time of Flight

Princip ToF snímače, jak název napovídá, je založen na měření času, za který paprsek urazí měřenou vzdálenost, přesněji vzdálenost tam a zpět. Snímač se dělí na dvě části - přijímač a vysílač. Z vysílače jsou vysílány světelné pulzy nebo souvislý paprsek světla. Ty jsou odraženy od objektu v prostoru zpátky do přijímače. Při použití pulzů je rozdíl času mezi odchozím a příchozím pulzem dobou letu paprsku. Pokud je použit souvislý paprsek, vzdálenost se počítá dle rozdílu fáze v příchozím a odchozím paprsku. Pro výpočet vzdálenosti d z doby letu t je použita úprava klasického vzorečku pro výpočet rychlosti. Za rychlost je dosazena rychlost světla c a vzdálenost je vydělena dvěma, protože paprsek letí dráhu tam a zpět:

$$d = \frac{ct}{2} \tag{3.1}$$

Důležitými parametry pro ToF snímače jsou vlnová délka, rozmezí měřitelných vzdáleností a přesnost. Různé vlnové délky mohou mít rozdílnou propustnost a odrazivost pro stejný povrch.



Obr. 3.1: Znázornění principu ToF senzoru [22]

Elektromechanické 3D LiDARy

Obvyklá konstrukce je z několika ToF snímačů, tvz. hlav. Hlavy jsou poskládány na sobě ve sloupci, který se otáčí o 360° ve vodorovné pozici, často o frekvenci *10 Hz* a více. Nejčastější jsou dva způsoby čtení dat ze snímače, buď po celé otáčce snímače, nebo souvislý proud dat během otáčení, kdy není třeba čekat na dokončení otáčky. Výhodou elektromechanických 3D LiDARů je velké zorné pole, omezené obvykle pouze směrem do výšky. Nevýhody pak plynou z konstrukce, snímač obsahuje pohyblivé části, kvůli kterým má snímač nižší životnost než solid-state LiDAR. Mechanická konstrukce má také vliv na vyšší váhu a větší velikost, a také velkou pořizovací cenu. Další nevýhodou může být malá přesnost ve vertikálním směru, tento problém je často řešen přidáním více hlav, ale tím se samozřejmě zvyšuje i cena.

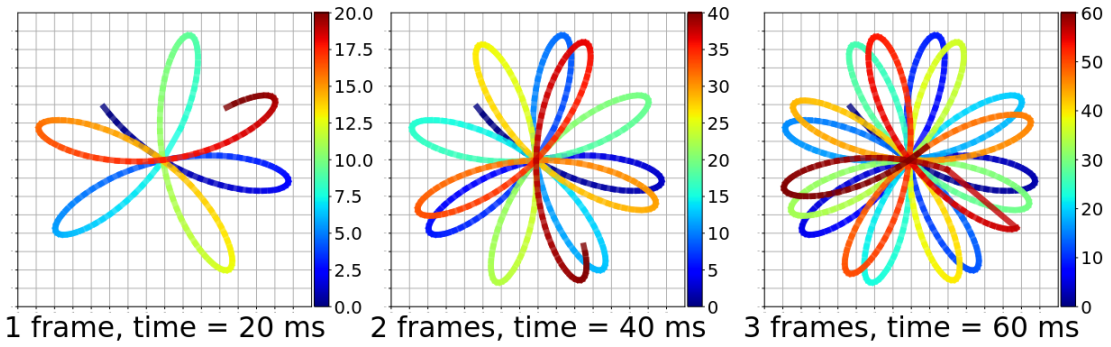
Solid-state LiDAR

Solid-state LiDAR má několik odlišností od elektromechanických LiDARů a výslednými daty spíše připomíná hloubkovou kameru. Snímač má pouze jednu laserovou hlavu, jejíž paprsek je natáčen pomocí MEMS elektroniky. Velkou výhodou oproti elektromechanickým LiDARům je malá velikost, větší životnost a mnohem nižší pořizovací cena. Tyto snímače mají menší zorné pole a kuželovitý tvar snímané oblasti, ale větší úhlové rozlišení, tedy snímají mnohem více bodů v prostoru. Navíc mají větší frekvenci snímání. Nevýhodou může být nepravidelný vzor snímaní dat, který je pro největší oblast pokrytí často řešen neopakující se dráhou snímání. Příklad nepravidelného vzoru snímání je na obrázku 3.2. Jelikož snímač má jen jednu laserovou hlavu, tak jsou jednotlivé body mnohem více zatíženy rozostřením kvůli pohybu a menší přesností na okraji snímaného prostoru[17]

Porovnání snímačů

Tabulky 3.1 a 3.2 porovnávají parametry elektromechanických 3D LiDARů. Počet řádků je počet hlav, které snímač obsahuje.

V tabulce 3.3 je uveden přehled parametrů solid state LiDARů, data jsou ze zdrojů [23], [24], [25] a [26], zdroje jsou seřazené stejně jako snímače v tabulce. Největším



Obr. 3.2: Znázornění trajektorie snímání solid-state LiDARu. Dráha je promítnuta na plochu metr před snímačem, čas vzorku je znázorněn barvou. [22]

Tab. 3.1: Přehled elektromechanických 3D LiDARů - Velodyne [27]

Výrobce	Velodyne				
Snímač	VLS-128-AP	HDL-64S2	HDL-32E	VLP-32c	VLP-16
Počet řádků	128	64	32	32	16
Frekvence [Hz]	5-20	5-20	5-20	5-20	5-20
Přesnost [m]	±0.03	±0.02	±0.02	±0.03	±0.03
Max vzdálenost [m]	300	120	100	200	100
Min vzdálenost [m]	N/A	3	2	1	1
Zorný úhel [°]	40	26.9	41.33	40	30
Vertikální rozlišení [°]	0.11	0.33	1.33	0.33	2.0
Horizontální rozlišení [°]	0.2	0.16	0.16	0.2	0.2
Vlnová délka [nm]	903	903	903	903	903
Váha [kg]	3.5	13.5	1.0	0.925	0.830

výrobcem, který pokrývá většinu trhu je Livox, který má hned několik výrobků. Nedávno byl představen firmou Intel LiDAR pro širokou veřejnost, cenově je ze všech nejpřístupnější navíc obsahuje RGB kameru a výpočetní software, ale cena se odráží na parametrech a jeho použití je limitováno pro vnitřní prostory.

Výběr LiDARu

Při výběru snímače byly preferovány rotující elektromechanické LiDARy. Oproti solid-state LiDARům mají výhodu, že nejsou takovou novinkou na trhu a pro jejich použití existuje více projektů řešících SLAM, protože pro solid-state LiDARy je adaptována pouze část existujících projektů. Při zúžení výběru LiDARů na pouze

Tab. 3.2: Přehled elektromechanických 3D LiDARů - Ostatní výrobci [27]

Výrobce	Hesai		Ouster		RoboSense
Snímač	Pandar64	Pandar40p	OS1-64	OS1-16	RS-LiDAR-32
Počet řádků	64	40	64	16	32
Frekvence [Hz]	10,20	10,20	10,20	10,20	5,10,20
Přesnost [m]	±0.02	±0.02	±0.03	±0.03	±0.03
Max vzdálenost [m]	200	200	120	120	200
Min vzdálenost [m]	0.3	0.3	0.8	0.8	0.4
Zorný úhel [°]	40	40	33.2	33.2	40
Vertikální rozlišení [°]	0.167	0.33	0.53	0.53	0.33
Horizontální rozlišení [°]	0.2	0.2	0.35	0.35	0.2
Vlnová délka [nm]	905	905	850	850	905
Váha [kg]	1.52	1.52	0.425	0.425	1.17

Tab. 3.3: Přehled parametrů solid-state 3D LiDARů

Výrobce	Intel	Livox			
Snímač	L515	AVIA	Mid-40	Mid-100	Mid-70
Frekvence [kHz]	23600	240	100	300	100
Přesnost [m]	±0.005	±0.02	±0.02	±0.02	±0.02
Max vzdálenost [m]	9	450	260	260	260
Zorný úhel [°]	70x55	70.4x77.2	38.4	98.4x38.4	70.4
Rozlišení [°]	N/A	0.05	0.05	0.05	0.1
Vlnová délka [nm]	860	905	905	905	905

elektromechanické bylo dále vycházeno z dostupnosti snímače. Z daných snímačů je k dispozici snímač Velodyne HDL-32E, který bude pro účely této práce využit.

3.2 Návrh softwaru

Po softwarové stránce je uvedeno, že projekt běžel při testování v distribuci ROSu Noetic a na operačním systému Ubuntu 20.04 [11].

Jelikož robotický systém ROS Noetic zaručuje podporu pouze systému Ubuntu 20.04, je jeho použití omezeno ze strany operačního systému. Protože tímto systémem používaný notebook nedisponuje nabízí se několik možností jak tuto podmínku splnit a projekt zrealizovat.

První, a po výpočetní stránce bez žádných omezení je nainstalovat Ubuntu 20.04 jako plnohodnotný operační systém, na druhou stranu tato možnost má několik úskalí z pohledu uživatele, kdy je potřeba zprovoznit nový operační systém a přenést své pracovní prostředí týkající se této práce na tento systém, a neřeší přenositelnost tohoto projektu.

Druhou možností je použití virtuálního stroje. Tato možnost by mohla umožnit běh na jakémkoliv operačním systému a zaručila by nezávislost omezenou pouze hardwarem. Nicméně tato možnost může být výpočetně náročná, už jen z důvodu běhu dvou operačních systémů zároveň. Další nevýhodou může být omezení hardwarových možností, například využití grafické karty. Proto běh náročných aplikací nemusí být vždy plynulý. Například by mohl nastat problém s programem RViz při zobrazování mračen bodů.

Další možností je využití částečné virtualizace pomocí Dockeru. Ten umožňuje větší nezávislost na operačním systému, jeho prostředí, nainstalovaných knihovnách a programech. Protože nevirtualizuje celý operační systém, není tak výpočetně náročný a nemá takové omezení jako virtuální stroj. Další výhodou je jeho oddělenost, a protože často obsahuje pouze nejnutnější aplikace pro běh, je výhodou i jeho malá velikost. Jednou z nevýhod může být horší zpřístupnění některých hardwarových prostředků, jako je například grafická karta, nicméně to se vybraného projektu netýká. Proto byl vybrán pro oživení tohoto projektu Docker.

3.2.1 Docker

Na rozdíl od virtuálního stroje Docker používá původní jádro operačního systému a virtualizuje pouze nástavbu na něm. Vytváří tvz. kontejner (*container*) v kterém jsou instalovány a spouštěny požadované aplikace.

Kontejner je virtualizované prostředí, ve kterém je možné spouštět aplikace. Je možné ho ovládat přes příkazovou řádku nebo pomocí programovatelného rozhraní. Pro vytvoření kontejneru je třeba vytvořit šablonu tvz. *image*.

Image je snapshot daného kontejneru, jsou v něm zachycena veškerá data o tom, jak a co daný kontejner obsahuje. Image je vytvářen podle textového souboru *Dockerfile*, případně z existujícího kontejneru. Každý nový image vychází z již existujícího image. Existuje oficiální repozitář, na kterém je uloženo velké množství image vycházejících z různých operačních systémů nebo konfigurujících různé aplikace a prostředí. Velmi často tak není třeba při vytváření řešit instalaci a stažení většiny potřebných aplikací, ale již pouze specifické závislosti.

Image tvoří jednotlivé vrstvy, které jsou definovány v Dockerfile. Vrstvy jsou stavěny vždy na vrstvách předchozích a image je mohou sdílet mezi sebou. Tedy pokud máme *image 1* a z něj vytvoříme *image 2* a *image 3*, v paměti bude uložen

image 1 pouze jednou, a pokud by *image 2 a 3* měly například první vrstvu stejnou, byla by taktéž sdílena. Kromě výhody menšího úložiště to umožňuje rychlejší sestavení image při přidání nové vrstvy, případně úpravy poslední, protože spodní vrstvy nejsou závislé na horních (naopak to nicméně neplatí) [28].

Návrh sestavení image

K vytvoření image pro tuto práci bude vycházeno z image *osrf/ros:noetic-desktop-full*. Pod image *ros* je vydáváno pouze několik kontejnerů, které obsahují různě obsáhlé verze ROSu, a proto je použit image *osrf/ros*, který nabízí a rozšiřuje původní nabídku image, a to včetně plnohodnotného image, který obsahuje plnou instalaci ROSu i s nástroji pro vývoj a vizualizaci dat [29]. Použitím tohoto kontejneru je zajištěno funkční prostředí pro ROS Noetic a není třeba vytvářet vlastní image vycházející z Ubuntu 20.04 a provádět v něm instalaci ROSu.

Následně v další vrstvě budou aktualizovány databáze aplikací a jejich aktualizací a nainstalovány potřebné nástroje pro vytvoření dalších vrstev, jako například git. V dalších vrstvách pak budou nainstalovány knihovny potřebné k funkčnosti projektu. V předposlední vrstvě budou podpůrné aplikace pro jednodušší vývoj a práci v kontejneru. V poslední vrstvě je v plánu instalovat a zkompilovat samotný projekt a při spouštění kontejneru zadat pouze příkaz ke spuštění projektu. Při práci na projektu a jeho ladění tato vrstva není v plánu pro jednodušší práci s kódem projektu.

3.2.2 Návrh v rámci ROS

Data ze snímače jsou publikována v rámci ROS2. Dodaná offline data jsou také uložena ve formě bagu podporujícího ROS2. Protože projekt podporuje pouze ROS1, vniká problém s přenosem dat ze snímače na vstup projektu. Pro offline data je možné využít dvě řešení a to převedení bag souboru do verze podporující ROS1, nebo použití bridge mezi ROS1 a ROS2 pro daný topic. Pro online data zbývá pouze varianta s použitím bridge.

Pro vyhodnocení offline dat a odladění algorimu na nich je v plánu pro jednodušší manipulaci navržen způsob první, tedy převedení do verze bag souboru podporujícího ROS1. Výhodou tohoto přístupu je získání více možností jak s daty pracovat, oproti ROS2 je možné v ROS1 přehrávání bag souboru pozastavit, krokovat jednotlivá data nebo spouštět data od určitého a do určitého času. Za další výhodu se dá považovat, že v této fázi není stále potřeba práce a instalace ROS2 a řešení problémů, které se kvůli tomu naskytnou. Následně před přechodem na online data bude na offline datech vyzkoušen a otestován bridge.

Pro vytvoření bridge je potřeba mít instalovaný ROS2. Protože celý projekt bude v dockeru, je v plánu zprovoznit bridge také v dockeru ve vlastním kontejneru. Výhodou je přenositelnost a nezávislost na operačním systému, protože stejně jako jednotlivé verze ROS1, tak i verze ROS2 jsou plně podporovány pouze na určitých verzích operačních systémů. Pro bridge mezi ROS1 a ROS2 je dostupný kontejner *osrf/ros:foxy-ros1-bridge*, který obsahuje obě verze ROSu včetně spustitelného balíčku, který vytváří bridge [29].

Bridge je možné zprovoznit ve dvou variantách, první tvz. dynamická umožňuje převod všech publikovaných zpráv mezi oběma variantami ROSu. Druhá varianta převádí pouze vybrané zprávy mezi verzemi. Protože projekt potřebuje na vstupu pouze jeden topic, je zvolena druhá možnost, kdy se bude pouze převádět zpráva s mračnem bodů ze snímače. Pro definici, který topic a jak převádět, je použit parametrický server ROS1. Ten se používá pro jednoduché sdílení parametrů v rámci ROSu. Proto musí být vytvořen yaml soubor obsahující definici a dodatečné parametry pro převáděný topic.

Pro spouštění projektu bude využito launch souboru, protože v projekt je třeba spustit několik node a definovat několik parametrů. Původní projekt již takový to soubor obsahuje, proto se bude při tvoření nového vycházet z něj. Budou vytvořeny dva launch soubory, jeden pro práci na offline datech převedených do ROS1, a druhý pro data vedená přes bridge, tedy online i offline data. Cílem je, aby pro spuštění projektu v ROSu bylo potřeba pouze jednoho nebo malého množství příkazů. Díky tomu pak bude možné jednoduché spuštění celého projektu.

4 Oživení projektu a testování na předem naměřených datech

Před oživením projektu je první třeba vytvořit kontejner v Dockeru. Následně byl zprovozněn projekt pro dodaná data. A nakonec byl algoritmus upraven na vnitřní prostředí a dodaná data. Dodané byli tři bag soubory, pořízené v přízemí budovy Technická 12 Fakulty elektrotechniky a komunikačních technologií. Dva záznamy jsou nasnímaná data chodby a třetí je v rámci jedné místnosti. Chodba není přespříliš členitá, nejčastějšími objekty jsou skříně na přezůvku, židle a dveře do místností, velkou část tedy tvoří rovné plochy. Naopak místnost by se dala charakterizovat jako velmi členitá, nachází se zde stoly s počítači, židle, skříně, regál, pohovka, různé vetší objekty na podlaze, ale i plno malých objektů na stolech a v regálu.

4.1 Vytvoření a spouštění kontejneru

Jak již bylo zmíněno v předchozí kapitole, pro vytvoření kontejneru je třeba vytvořit image pomocí Dockerfile. Image bude vycházet ze zmíněného *osrf/ros:noetic-desktop-full*, který zaručuje funkční instalaci ROSu se všemi jeho součástmi. Rozdělení vrstev na tuto práci nemá velmi velký vliv, stačilo by vše ponechat pouze v jedné vrstvě. Případně mít pouze poslední vrstvu do které je možné přidávat aplikace, které se ukázali až v průběhu času jako potřebné. Ale samotným Dockerem je doporučovaná praxe rozdělovat na více vrstev. A ve výsledku to může přinést pouze pozitiva. Obecně při tvoření image to může poskytnout výhodu kratšího času na sestavení a menší zatížení na úložiště, při více verzích image. Potenciální výhodou v průběhu práce může být urychlení času pokud by například došlo ke zjištění, že některá z knihoven je ve špatné verzi nebo špatně nainstalovaná. Obecně tento přístup usnadňuje tvorbu a úpravy image.

V první vrstvě je aktualizován list balíčků a aplikací v systému. Aplikace nejsou aktualizovány na novější verze, pouze je synchronizován list verzí instalovaných aplikací s vzdálenými repozitáři, ze kterých se aplikace následně instalují. Obvykle jsou aktualizovány informace pouze o oficiálních repozitářích, jejichž bezpečnost a kompatibilita je ověřena samotnými lidmi stojícími za daným operačním systémem. Pro tento krok jsou aktualizovány i informace o přidaném repozitáři z kterého je možné instalovat a aktualizovat ROS a jeho součásti, jelikož kontejner informace o tomto repozitáři obsahuje a je to doporučená cesta jak ROS instalovat a aktualizovat. Toto je nutný krok pro možnost instalace aplikací přes příkaz *apt-get*. V té samé vrstvě je instalován *git*, jakožto nástroj bez kterého nejde stáhnout některé knihovny.

První z potřebných knihoven, která je instalována je knihovna *Ceres Solver*, jejíž instalace je rozdělena do dvou vrstev. Tato knihovna je pro výpočty a matematickou optimalizaci. Protože je knihovna instalována z kódu jsou první nainstalovány knihovny, na niž je závislá. Ty jsou nainstalovány za pomoci *apt-get*, který mimo jiné řeší a instaluje i potřebné závislosti knihoven. V další vrstvě je nainstalována samotná knihovna *Ceres Solver*, její kód je stažen za pomoci *gitu* a nainstalován pomocí programů *cmake* a *make*. V dalších dvou následujících vrstvách jsou nainstalovány knihovny PCL, která umožňuje práci s mračny bodů a GTSAM, která je využita pro graf SLAMu. Obě jsou instalovány za pomoci *apt-get*, pro GTSAM je pouze třeba přidat do seznamu repozitářů jeho vlastní. Následující vrstva instaluje *hector trajectory* jakožto jediný balíček ROSu. Tento balíček zajišťuje vizualizaci a umožňuje zaznamenávání trajektorie. Nakonec jsou instalovány nástroje pro jednodušší vývoj. Prvně zde bylo pouze vytvoření složky pro ukládání dat z algoritmu, a postupně se přidala instalace aplikace *gdb*, pro debugování programu a aplikace *psmisc*, která dodává příkazy pro správu procesů.

Samotný image je pak sestrojen pomocí příkazu:

```
docker build -t dev_scfloam:1 .
```

Argument *-t* a následný název s dvojtečkou označují takzvaný tag, tedy pojmenování daného image. Image může mít několik různých pojmenování, název za dvojtečkou se obvykle používá pro odlišení verze, často používané je slovo *latest*, které označuje verzi, která bude použita pokud nic za dvojtečkou není specifikováno. Tečka na konci označuje cestu ke složce obsahující Dockerfile. V tomto případě je příkaz spouštěn ze složky která obsahuje Dockerfile, proto může být použita tečka jako zástup momentální systémové cesty.

Jelikož spouštění kontejneru samotného je často doprovázeno velmi dlouhý příkazem, byl proto vytvořen skript *run_docker.sh*, který obsahuje pouze příkaz na spuštění kontejneru, a jako jediný argument je název image. První argument příkazu pro spuštění je *-it*, který propojí kontejner s bash terminálem, čímž při ukončení bash terminálu bude ukončen i samotný kontejner. Následně je použit argument, který zvětšuje privilegia danému kontejneru a umožňuje v kontejneru provádět skoro všechny věci, které jsou možné na původním systému. Poté je přidán příkaz, který sjednocuje síťové vlastnosti dockeru s původním systémem a zjednodušuje práci se síťovými prvky. Následující sada argumentů je přidána pro zprovoznění zobrazovacího serveru původního systému, to umožňuje spouštění grafických rozhraní aplikací. Existuje několik variant jak toho dosáhnout, byla zvolena nejjednoduší varianta, u které je nutné přes příkaz:

```
xhost + local:docker
```

povolit přístup Dockeru k zobrazovacímu serveru. Při hledání jak spustit grafické rozhraní bylo vycházeno ze zdroje [30].

Následují příkazy pro připojení úložišť z původního systému. Jedná se o složku obsahující pořízená data, kam se při běhu budou také ukládat výsledky systému. Dále je připojen samotný kód, čímž bude jednodušší úprava a práce s kódem a zároveň nebude nutné před každým ukončením kontejneru změny v kódu uložit na vzdálené úložiště. Nakonec je připojen soubor *.bashrs*, který konfiguruje bash konzoli a kromě příkazu

```
source ~/catkin_ws/devel/setup.bash
```

který konfiguruje prostředí pro ROS, je připojení toho souboru pro tuto práci nepodstatné. Nakonec je při spuštění kontejneru spuštěn příkaz, který zkompiluje kód a spustí bash konzoli.

4.2 Oživení projektu

Pro spuštění projektu na dodaných datech, bylo prvně nutno převést data do správného formátu pro ROS1. Při oživování projektu byl zároveň zkoušen i jeden KITTI dataset na jehož spuštění byl projekt připraven [19]. To umožnilo nejen rychlejší ověření funkčnosti před většími úpravami projektu na dodaných datech, ale i porovnání pro hledání důvodů nefunkčnosti pro dodaná data a ověření případných úprav projektu, zdali funguje vše jak má po stránce spuštění a běhu programu.

4.2.1 Konverze dat mezi ROS1 a ROS2

Jelikož byly data dodána v bagu ROS2, bylo je třeba konvertovat do bag souboru pro ROS1. K tomu posloužil nástroj *Rosbags*, který poskytuje funkce pro práci s bag soubory, mimo jiné převod mezi jejich verzemi v ROS1 a ROS2. Pro funkci tohoto nástroje není třeba instalace ani jedné z obou verzí ROSu. Jediné závislosti, které jsou potřeba, je Python a jeho správce balíčků *pip*, které má většina linuxových systémů již nainstalované [31]. Protože bylo třeba nástroje jen jednou a není závislý na ROSu byl nainstalován na původním systému mimo kontejner. Data byla převedena pomocí příkazu:

```
rosbags-convert input_bag
```

kde za *input_bag* byl dosazen název bag souboru.

4.2.2 Úprava kódu a jeho spuštění

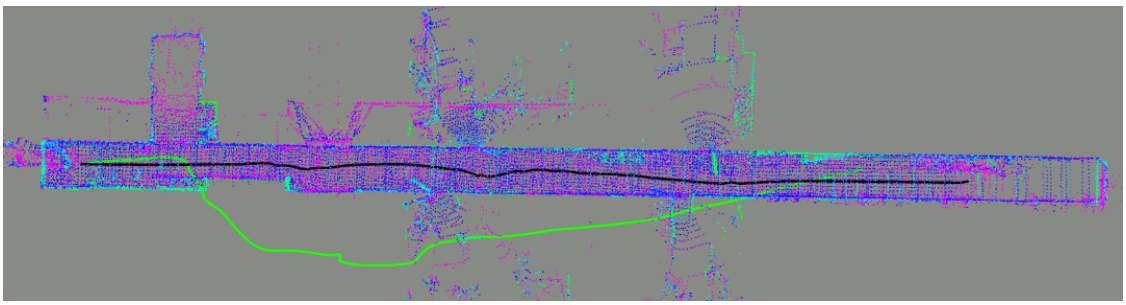
V této části budou popsány úpravy kódu, které byli potřebné ke spuštění, ale i úpravy, které usnadnili práci a ladění algoritmu. Jak již bylo zmíněno, prvně byl zprovozněn projekt na datasetu KITTI. Jedním z důvodů bylo, že prvotní pokusy o spuštění na dodaných datech skončily neúspěchem a vyhazovaly výjimku knihovny pro práci s mračny bodů. Proto bylo přistoupeno k variantě zkusit spustit projekt na datasetu, na který byl upraven. Jediné, co bylo třeba pro spuštění KITTI datasetu, byla změna cesty pro ukládání dat. To vedlo k závěru, že problém je způsoben nějakým rozdílem v dodaných datech. Problémem bylo, že poslední node *sc_floam_laserLO* padá na výjimce vyvolané při ukládání jednoho ze souborů. Podstatným rozdílem mezi datasety se ukázala být pravdivá trasa, po které se snímač pohybuje, a která se v dodaných datasetech nenacházela. Důvodem byla implementace node, která nebyla vytvořena pro jiné datasety, které neobsahují tuto informaci. Ale z podstaty algoritmu a projektu samotného vyplývá, že odstranění záznamu pravdivé cesty by nemělo mít vliv na funkčnost projektu. Výsledkem byla úprava jedné z hlavních funkcí, která spouští konstrukci a optimalizaci grafu. V této funkci byla informace o pravdivé pozici použita pouze pro časovou synchronizaci s ostatními daty. Díky tomu se při jejím nedodání, i pokud nebyla ukládána velká část výpočtů, nespouštěla. Pro zprovoznění byly podmínky, které se vázaly na data pravdivé pozice, odděleny od ostatních podmínek. Výsledný kód pak umožňuje fungování pro původní dataset, který obsahuje informaci o pravdivé pozici, tak i pro dataset, který tuto informaci neobsahuje.

Další změnou v kódu bylo vytvoření makra pro jméno pozice transformace topicu */tf*. Důvodem pro tuto změnu byl stejný název jak pro pozici v dodaných datasetech, tak pro daný algoritmus. Při změně jména bylo pro jednodušší úpravu v budoucnu vytvořeno v souborech, kterých se tato změna týká, preprocesorové makro, které případnou další změnu jména zjednoduší.

Při několika testech se vyskytly další problémy s ukládáním dat, tentokrát s ukládáním výsledné mapy. Problémy se nevyskytovali vždy ale pouze občas v závislosti na nastavených parametrech. Node *sc_floam_laserLO* po spuštění vyhodilo výjimku, že nemá data pro uložení. Důvodem byla podmínka alespoň dvou pozic na mapě, při menším počtu se mapa nevytvářela a vracela se prázdná. Protože se mapa aktualizuje a vytváří jednou za 10 sekund, byla většinou tato podmínka splněna. Problém nastával pokud robot stál na počátečním místě déle nebo byl změněn čas, za který se mapa vytváří. Řešením byla změna počtu potřebných pozic na postačující dvě pozice. Poté se již tento problém při testech na dodaných datasetech neobjevoval.

Poslední úpravy kódu se týkali přidání možnosti zadat hodnoty parametrů, které

byli často upravovány v launch souboru. Výhodou zadávání v tomto souboru je odstranění nutnosti kompilace celého programu při jejich změně a možnost je zadat i při spouštění launch souboru. Do těchto parametrů patří cesta ke složce pro ukládání souborů a několik parametrů rozlišení. Při zadávání cesty bylo navíc ošetřeno, aby byla daná složka vytvořena, pokud neexistuje, a to včetně celé její systémové cesty. Druhým parametrem je */map_resolutionLO*, který určuje rozlišení mapy pro hledání smyček a pro pozice s jejich body v grafu. Poslední parametr je parametr pro rozlišení výsledné mapy */mapviz_filter_size*.



Obr. 4.1: Porovnání trajektorie z vnitřní odometrie robota, znázorněna zelenou, a výsledné cesty algoritmu, znázorněna černou.

Další úpravou v rámci launch souboru byla změna node *word2map_tf*, aby pro jednotlivá data, byl stejný počátek jako vnitřní odometrie robota. Po porovnání bylo zjištěno, že tato odometrie není nejpřesnější, a že vybraný projekt dosahuje mnohem větší přesnosti. Porovnání lze vidět na obrázku 4.1. Dále proto s těmito daty nebylo pracováno. Po těchto úpravách je kód spustitelný a po prvních spuštěních na datech jsou od pohledu vytvářeny správné mapy, ale s malým počtem bodů.

4.3 Ladění algoritmu

První změnou bylo dosazení správných parametrů snímače do algoritmu. Protože algoritmus je odladěný na snímačích Velodyne, nebylo třeba měnit mnoho parametrů souvisejících se snímačem. Mezi parametry týkající se snímače patřily:

- *scan_line* určuje počet řádků snímače,
- *scan_period* je perioda, za kterou přichází jeden snímek, tento parametr nebyl změněn,
- *max_dis* popisuje maximální vzdálenost, kterou je snímač schopen snímat,
- *min_dis* popisuje minimální vzdálenost, kterou snímač je schopen snímat,
- *LiDAR_HEIGHT* určuje výšku LiDARu aby mohl být výsledek dosazen do správných souřadnic, na funkčnost algoritmu jinak nemá vliv.

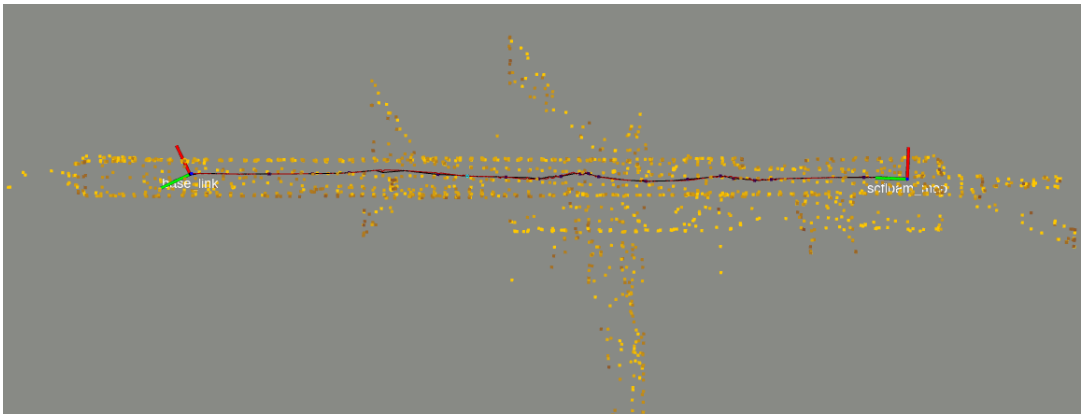
LiDAR_HEIGHT je jediný parametr, který není definovatelný v launch souboru, ale v kódu. Ještě je možné změnit parametr *vertical_angle* a přidat další parametry snímače, ale po vyhledání jejich vlivu v kódu bylo zjištěno, že nemají na daný algoritmus žádný vliv.

Následující proměnné byly postupně upravovány, aby výsledná mapa dosahovala dostatečně dobrého výsledku. Filtrování bodů je prováděno pomocí filtru, který oblast rozdělí do voxelů, a výsledkem je aproximované těžiště těchto voxelů. Více o tomto filtru zde [32]. Většina hodnot, od kterých se odvíjelo ladění algoritmu, byla odhadnuta na základě přihlédnutí k rozměrům dodaných datasetů, protože původní hodnoty byly pro ně přestřelené. Upravovanými proměnnými jsou:

- *map_resolution* je parametr určující velikost filtru pro mračno bodů pro odometrii a výslednou mapy z odometrie. Tento parametr byl původně nastaven na hodnotu 40 *cm*, která se může zdát pro orientaci ve vnitřním prostředí jako velká, proto byla postupně snížena na hodnotu 15 *cm*, nižší hodnoty již nepřispívali k přesnosti, ale naopak algoritmus se častěji ztrácel a vracel špatné výsledky, zvláště pak pro záznam z místnosti.
- *map_resolutionLO* určuje velikost filtru pro body, které jsou použity pro výpočet a hledání smyček. Jako u předchozího rozlišení, i tento parametr byl nastaven na 40 *cm*, podobnou logikou se došlo k výsledným 15 *cm* pro dodaná data.
- *mapviz_filter_size* je velikost filtru, který je použit před publikováním a uložením výsledné mapy. Jelikož má tento parametr vliv pouze na výslednou mapu a výpočetní náročnost byl, původně nastaven na hodnotu 60 *cm*, která může být pro popis vnitřních prostorů značně nedostatečná. Přestože je mapa tvořena již z bodů filtrovaných v předchozím kroku, je možné použití i menší hodnoty než v předchozím kroku. V předchozím kroku jsou body filtrovány a připisovány jednotlivým pozicím grafu, a následně je z kombinace bodů těchto pozic tvořena výsledná mapa, která je poté filtrována. Následkem menší velikosti filtru je získání většího počtu bodů v oblastech, na které je viděno z více pozic. Okraje výsledné mapy budou obsahovat méně bodů, protože se zde nepřekrývá tolik bodů z různých pozic. Proto je výsledná nastavená hodnota 10 centimetrů.
- *keyframe_meter_gap* je vzdálenost, o kterou se musí robot posunout aby byla vytvořena nová pozice v grafu. Původní hodnota byla 2 metry, to působí jako velká hodnota pro místnost s největším rozměrem okolo 10 *m* Hodnota byla postupně upravena na 40 *cm*.
- *keyframe_deg_gap* je úhel natočení, o který se musí robot otočit, aby byla vytvořena nová pozice v grafu. Pro vytvoření pozice je potřeba splnění buď vzdálenosti, nebo natočení. Protože tato hodnota nezávisí na velikosti rozměrů,

byla nezměněna.

- *sc_max_radius* je parametr určující maximální hranici pro deskriptor Scan-Context. Tento parametr byl původně 80 metrů, ale v komentáři byly doporučeny hodnoty 20 a 30 metrů pro vnitřní prostory, protože data z místnosti přesahují vzdálenost 20 metrů jen na bodech, které jsou snímané přes okno a na protější zdi, a protože chodba je dlouhá 43 metrů byla zvolena jako adekvátní možnost 20 metrů, dále tento parametr nebyl upravován, protože určuje pouze maximální vzdálenost, kdy tato vzdálenost je jinak určena dynamicky.



Obr. 4.2: První výsledek mapy před úpravou parametrů s vyznačenou trajektorií.

Před změnami těchto parametrů a pouze úpravou parametrů snímače vypadala mapa jako na obrázku 4.2. Na obrázku se nachází data z prvního průjezdu chodbou, barva bodů znázorňuje intenzitu, kdy nejsvětlejší body ji mají nejvyšší, pozice *scfloam_map* je počátek cesty a *base_link* je pozice robota na konci, i přes špatně rozeznatelný rozdíl na tomto obrázku, červená cesta je pouze na základě odometrie a černá je výsledná cesta. Algoritmus i bez ladění vrací mapu, která na první pohled vypadá jako daná chodba. Což znamená, že je schopen odhadovat odometrii, která není daleko od pravdy. Pravdivost cesty nebylo jak ověřit, pouze trajektorie v ose Z neustále stoupala. To jde vidět na obrázku 4.3 kde je zobrazena závislost výšky na vypočítané vzdálenosti uražené ve zbylých osách. Nejzřetelnějším problémem je malý počet bodů, kdy na obrázku je přibližně 1800 bodů. Ač z těchto bodů je možné určit základní rozměry místnosti, tak například otevřené dveře jsou reprezentovány pouze několika body.

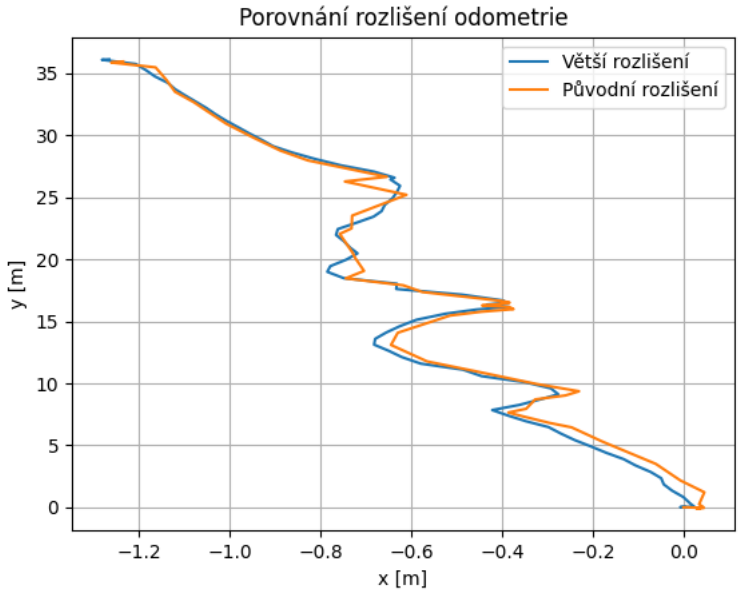
Prvním a nejvíce laděným parametrem bylo právě rozlišení pro získání nejen podrobnější mapy, ale i přesnější odometrie. Pro rozlišení bodů jdoucí do odometrie byla snaha mít co nejmenší rozlišení pro co možná nejpřesnější určení změny polohy. Postupovalo se od hodnoty filtru 20 *cm* níž, ale u hodnoty 10 *cm* odometrie nefungovala tak, jak by se dalo předpokládat. V místnosti, která byla velmi členitá se

Obr. 4.3: Závislost výšky na vzdálenosti ujeté v osách X a Y.

v některých momentech odometrie začala špatně vyhodnocovat, nejzřetelnější bylo špatné vyhodnocení rotace. To ukazuje, že menší filtrací algoritmus získává na přesnosti, kterou lze pozorovat i bez znalosti, ale naopak ztrácí robustnost a může být malými členitými předměty špatně ovlivněn. Na obrázku 4.4 je možné vidět porovnání výsledku odometrie, tedy bez hledání smyček, pro různé velikosti filtru. Je možné pozorovat, že odometrie s větším rozlišením má hladší průběh. Sice je na daném obrázku je možné pozorovat detailnější změny jen v rámci osy x, nicméně o to více jsou vidět. Podstatné bylo najít hodnotu, která bude dostatečně robustní a dostatečně přesná pro průběh na daných datech.

Další parametr, který se upravoval je rozlišení pro konstrukci a optimalizaci grafu a hledání smyček. Při úpravách tohoto algoritmu jsem vycházel z podobné hodnoty jako pro rozlišení mapy. Menší filtr zde pomáhá vytvořit více jedinečnou oblast, tedy čím větší filtr je, tím je větší šance, že se bude v podobných prostorách vytvořený deskriptor opakovat. Nicméně naopak velmi malý filtr může zahrnout tolik detailů, že i dva skeny z velmi blízké pozice nebudou rozpoznány jako stejné. Hledání správné hodnoty probíhalo podobným způsobem jako u předchozího parametru a nakonec bylo ponecháno na stejné hodnotě 15 cm.

Pro parametry, které určují vznik nové pozice v grafu, je nejlepší hodnotou ta nejmenší. Nicméně velmi mnoho pozic může znehodnotit fungování algoritmu velkou výpočetní náročností. Druhou zátěží, která se může stát, je časté nacházení smyček. Tyto smyčky mohou být velmi blízko sebe a nemají pro zlepšení přesnosti velký vliv



Obr. 4.4: Porovnání rozlišení na výsledku odometrie, počátek os se nachází v počáteční pozici robota, velikost filtru pro původní rozlišení je 40 *cm* a pro větší je hodnota filtru 15 *cm*.

a pouze jejich výpočet navíc zvětšuje výpočetní náročnost.

5 Přenesení na robota a testování na něm

Pro přenesení projektu na reálného robota bylo prvně třeba zprovoznit brigde mezi ROS2 a ROS1. Doladit algoritmus, aby byl více robustním, a vyřešit problémy, které se do teď neprojevily. Následně byli zaznamenány výsledky algoritmu a vstupní data. Tato data slouží k porovnání přesnosti algoritmu. Dolaďování algoritmu oproti offline datům bylo minimální. Algoritmus byl funkční a vracel, skoro již od počátku testů, uspokojivé výsledky. Jedinou změnou bylo zmenšení úhlu pro vytvoření nové pozice grafu. Důvodem proto bylo, že při vyšších rychlostech otáčení se odometrie robota občas ztratila. Tato změna zlepšila stabilitu při častém otáčení, případně jízdě v kruzích.

Při vytvoření image bylo nakonec odstoupeno od návrhu přidat do kontejneru i kód této práce. Bylo vyhodnoceno, že vytvoření toho image nepřinese mnoho ulehčení při práci, ale spíše při pozdějších drobných úpravách může práci udělat složitější, proto byl pro práci s online daty použit stejný image jako při práci s offline daty.

5.1 ROS bridge

Funkčnost bridge byla před testem na robotovi prvně otestována na offline datech. Při tomto testu, který byl pouze mířen na funkčnost mostu byl převáděn pouze topic s daty snímače. Pro topic byla zvolena velikost fronty, která obsahuje poslední publikovaná data. Velikost byla zvolena 5, aby se předešlo ztrátám dat pro algoritmus samotný, nicméně hlavním důvodem byl záznam výsledků do bag souboru. Pokud by byl chvilkově počítač více zatížen a nestihl by data zpracovat, mohlo by dojít k jejich ztrátě. Protože to nemělo viditelný vliv na zátěž paměti, hodnota dále nebyla nijak upravována a řešena. Tato data je třeba je načíst v ROS1 a to příkazem:

```
rosparam load bridge.yaml
```

kdy *bridge.yaml* je soubor obsahující tato data.

Bridge samotný se spouštěl ve svém vlastním kontejneru. Po otevření kontejneru stačilo nastavit proměnnou prostředí, která obsahuje adresu ROS Master z ROS1. Pro spuštění mostu pak stačí spustit jeden příkaz. Proměnnou prostředí je možné nastavit při spouštění kontejneru a příkaz je možné zadat při spuštění, proto je ve výsledku potřeba jen jeden příkaz na spuštění celého mostu:

```
docker run -it --net=host \
-e ROS_MASTER_URI=http://localhost:11311/ \
osrf/ros:foxy-ros1-bridge \
ros2 run ros1_bridge parameter_bridge
```

Příkaz je rozdělen zpětnými lomítky na více řádků. Argument *-e* nastavuje proměnnou prostředí, na dalším řádku je image a na posledním je příkaz pro spuštění bridge.

5.2 Úpravy oproti verzi pro offline data

Samotný algoritmus se potýkal, při prvních spuštění potýkal jen s jedním problémem, který byl nutný řešit změnou kódu, následně byla vytvořen vlastní launch soubor a doladěn jeden parametr.

5.2.1 Změny v kódu

Problém, který při testování vznikl, byl stejný jako jeden z problémů pro testování na offline datech. Node *sc_floam_laserLO* se ukončovala pokud se robot od začátku nehýbal. Node se ukončovala s výjimkou, že nelze uložit prázdná data výsledné mapy. Problém vznikal nedostatkem pozic, když vznikla pouze jedna pozice, proto ani již upravená podmínka nestačila. Nakonec bylo přistoupeno k vytvoření podmínky ve funkci, která ukládá a publikuje konečnou mapu. Podmínka ověřuje, zdali je mapa prázdná nebo ne, a pokud ano, je ukládání a publikace přeskočena. Jelikož publikace a ukládání mapy nemá s průběhem samotných výpočtů nic společného, a protože po úpravě nedocházelo k žádným změnám v průběhu a výsledcích, je předpokládáno, že problém mála pozic je významný pouze pro tuto část kódu, která ukládá a publikuje finální mapu. Pravděpodobně bylo opomenuto, nebo předpokládáno že situace s jednou pozicí po 10 sekundách běhu nenastane.

5.2.2 Launch soubor a parametry

V rámci tohoto souboru nenastalo mnoho změn, pouze bylo odebráno spouštění dat z bag souboru. Byla změněna proměnná, která říká, aby byl použit čas ze přehrávaných dat a ignorován reálný čas. Proměnná byla změněna na negativní hodnotu.

6 Zhodnocení výsledků

V této části budou popsány a zhodnoceny výsledky algoritmu jak pro data ze záznamu, tak pro data získaná na robotu v reálném čase. Výsledná data jsou pořízena algoritmem upraveným po ladění na online datech. Získané mapy budou porovnány s dodaným plánem části přízemí budovy Technická 12 Fakulty elektrotechniky a komunikačních technologií. Detailní obrázky celých map s trasami a s porovnáním k půdorysu se nachází v příloze.

6.1 Metody hodnocení

Vyhodnocení získaných map je z velké části zaměřeno na výsledný půdorys. Z části proto, že je dodán půdorys budovy jehož hodnoty jsou brány jako pravdivé, a z druhé části proto, že není možné ověřit a porovnat pravdivost nalezené cesty. U výsledné cesty je možné hodnotit data v rámci osy Z, protože se veškeré odchýlení od nuly dá považovat za chybu, při předpokladu, že se podlaha nachází v jedné rovině. Pro srovnání bude porovnána mapa při promítnutí na plán budovy. Z toho se bude hledat a vyvozovat odchýlení v rámci rotace a případně rozměrů. To bude použito pro data obsahující chodbu, protože na počátku trasy se data zarovnají s plánem na základě stěn, na konci trasy bude možné pozorovat případné změny. U map vygenerovaných z jedné místnosti to není tolik vhodné, protože k zarovnání jsou použity stěny dané místnosti a již se nemůžou použít pro zjištění zkreslení. Parametry, které se budou měřit budou představeny v následujícím seznamu:

- Vyhodnoceny budou vzdálenosti stěn na různých místech mapy v přibližné výšce 40 *cm* od podlahy mapy. U každé mapy bude místo měření specifikováno. Pro vyhodnocení bude použit výpočet absolutní a relativní chyby. Vzorec pro absolutní chybu je

$$\Delta = |x_m - x|\,, \tag{6.1}$$

  kde x_m je naměřená hodnota a x je správná hodnota z plánu budovy. Vzorec pro relativní chybu v procentech je

$$\delta = 100\% \times \frac{\Delta}{|x|}. \tag{6.2}$$

- Pokud dojde pro data z chodby k viditelnému odchýlení stěn od půdorysu, bude změřen úhel mezi mapou a půdorysem. Bod, ke kterému bude úhel vztažen, je bodem kdy se naměřená stěna začne odchylovat, nemusí se jednat o bod blízko počátku trasy.
- Pro vyhodnocení v rámci výšky bude změřen úhel mezi podlahou a stěnami, jako pravdivá hodnota se bude předpokládat pravý úhel.

- Porovnání dat s plánem půdorysu budovy.
- Protože se dá předpokládat, že rovina podlahy je stejná, jedno z kritérií, které bude posuzováno, je jak se mění výška během trasy robota, v ideální případě by měla být konstantní nebo pouze s minimální změnou.

6.2 Offline data

Pro testování na offline datech byly dodány dva záznamy chodby (*hall_pass_1, hall_pass_2*) a jeden záznam místnosti(*lab_pass_1*).

6.2.1 Měření vzdáleností

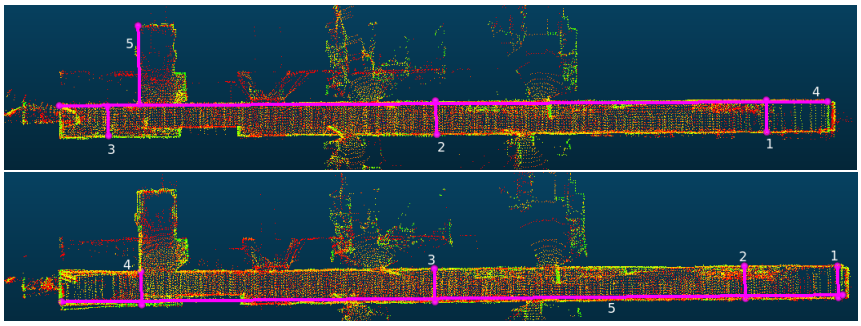
Na obrázku 6.1 jsou zobrazené úseky, ve kterých bylo provedeno měření bodů. Jednotlivé úseky pro měření, zvláště napříč byly vybrány u zdí, aby byla zachována linie. U druhého průjezdu byla změněna strana zdi, kde bylo provedeno měření, protože roh zakrývaly otevřené dveře.

Jelikož trasa pro první průjezd začíná na pravé straně obrázku, jsou jednotlivé délky napříč chodbou pojmenovávány na začátku, uprostřed a na konci trasy. Výsledná data jsou v tabulce 6.1, aritmetický průměr je $1,56\ cm$. Největší chyba je na datech, která jsou brána i z větší vzdálenosti od robota. Protože trasa robota končí necelých osm metrů před koncem chodby, je možné tento rozměr spolu s délkou přilehlé chodbičky brát jako větší vzdálenost, na kterou nebylo nasnímáno velké množství dat, a jsou zaznamenány pouze v menším počtu pozic grafu. Co se týče dat v blízkosti robota, největší chyba je $1,5\ cm$.

Tab. 6.1: Vzdálenosti z prvního průjezdu chodbou

	x[m]	x_m[m]	Δ[m]	δ[%]
Šířka začátek trasy	1,8	1,811	0,011	0,61
Šířka uprostřed trasy	1,8	1,815	0,015	0,83
Šířka konec trasy	1,8	1,789	0,011	0,61
Délka	43,2	43,176	0,024	0,05
Chodbička	4,4	4,417	0,017	0,39
Aritmetický průměr			0,0156	0,500

Měření z druhého průjezdu chodbou v opačném směru jsou v tabulce 6.2. Měření v chodbičce nebylo provedeno pro nedostatek bodů a pro rozdílný záznam dat v několika pozicích, jak je na obrázku viditelné. Při měření šířky chodby jsou data podobná jako pro předchozí průjezd. Největší rozdíl je v datech při měření délky.



Obr. 6.1: Měření vzdáleností pro oba průjezdy chodbou. Na horní obrázku je první průjezd - 1 - šířka začátek trasy, 2 - šířka uprostřed trasy, 3 - šířka konec trasy, 4 - délka, 5 - chodbička. Na spodní obrázku je druhý průjezd - 1 - šířka před začátkem trasy, 2 - šířka začátek trasy, 3 - šířka uprostřed trasy, 4 - šířka konec trasy, 5 - délka.

Hlavním důvodem je pravděpodobně to, že v některém bodě je stěna vyhodnocena v jiné poloze než v jiných, z důvodu špatné odometrie. Nejenže je špatná odometrie, která může být později zpřesněna optimalizací, ale body stěny z různých skenů nejsou ve stejné rovině, a to ubírá na přesnosti celému algoritmu. Pokud by robot jel dál do konce mapy, je pravděpodobné, že by se data zpřesnila.

Tab. 6.2: Vzdálenosti z druhého průjezdu chodbou

	x[m]	x_m[m]	Δ[m]	δ[%]
Šířka před začátkem	1,8	1,807	0,007	0,39
Šířka začátek trasy	1,8	1,784	0,016	0,89
Šířka uprostřed trasy	1,8	1,786	0,014	0,78
Šířka konec trasy	1,8	1,815	0,015	0,83
Délka	43,2	43,309	0,109	0,25
Aritmetický průměr			0,0322	0,628

Data místnosti z tabulky 6.3 jsou změřené rozměry přes střed místnosti, nezávisle na stejné výšce. Jelikož je místnost velmi členitá , je pouze malé množství bodů, které reprezentují stěny a jsou naproti sobě, a tyto body jsou pouze na některých místech. I přes použití filtrů s velikostí 10 *cm* a 15 *cm* je algoritmus přesný v řádu centimetrů, zvláště pro body, které jsou viděny z více pozic. Pouze pro jedno měření byla chyba větší něž 10 *cm* Pro body na okraji trasy, nebo body, které jsou viděny pouze na chvíli, je přesnost menší.

Tab. 6.3: Naměřené rozměry místnosti

	x[m]	x_m[m]	Δ[m]	δ[%]
Délka	10,7	10,686	0,014	0,13
Šířka	7,2	7,297	0,097	1,34
Aritmetický průměr			0,0555	0,739

6.2.2 Měření úhlů

Měření úhlů je provedeno v řezech, které jsou ve stejném místě jako měření na obrázku 6.1. Pro každý řez jsou změřeny dva úhly mezi stěnami a podlahou. První se vždy nachází po pravé straně robota nebo blíže k počátku trasy. Pro druhý průjezd chodbou nebyly změřeny úhly před začátkem trasy pro nedostatek bodů podlahy. Chyby hodnot úhlů se pohybují přibližně v rozmezí podobných hodnot. Při úhlech chodbičky hrál roli, kromě vzdálenosti, i menší počet bodů. Při menším počtu bodů se úhly měřily hůře, protože roviny podlahy a stěny se obtížněji určovaly. Kromě dvou hodnot, které se pohybovaly kolem 3°, ostatní hodnoty nepřesáhly 2°.

Tab. 6.4: Kolmost stěn v řezech při prvním průjezdu chodbou, φ_{m1} je úhel po pravé straně trasy nebo blíže startu, φ_{m2} je úhle po levé straně trasy nebo vzdálenější úhel od startu

	φ_{m1}[°]	Δ[°]	δ[%]	φ_{m2}[°]	Δ[°]	δ[%]
Šířka začátek trasy	90,86	0,86	0,96	86,66	3,34	3,71
Šířka uprostřed trasy	89,69	0,31	0,34	90,51	0,51	0,57
Šířka konec trasy	90,41	0,41	0,46	88,49	1,51	1,67
Délka	89,79	0,21	0,23	90,12	0,12	0,13
Chodbička	91,76	1,76	3,09	87,22	2,78	1,96
Aritmetický průměr					1,181	1,312

6.2.3 Porovnání s půdorysem

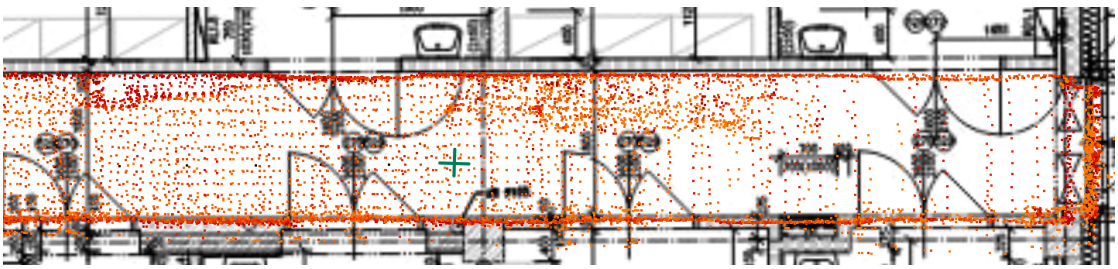
Výsledné mapy byly promítnuty na půdorys budovy, to je možné vidět na obrázcích B.1B.2B.4. Při porovnání mapy místnosti s půdorysem je možné vidět na levé straně ve spodní části přesah bodů přes zeď, dosahujících v některých místech 10 *cm*. V dané části místnosti se nachází otevřený regál s různými předměty, samotná plocha zdi je snímána jen částečně a nesouvisle. O to horší je i kombinování bodů z více pozic. Na pravé straně se nachází vysoká skříň, ta koresponduje s místem na plánku,

Tab. 6.5: Kolmost stěn v řezech při druhém průjezdu chodbou, φ_{m1} je úhel po pravé straně trasy nebo blíže startu, φ_{m2} je úhle po levé straně trasy nebo vzdálenější úhel od startu

	$\varphi_{m1}[°]$	$\Delta[°]$	$\delta[\%]$	$\varphi_{m2}[°]$	$\Delta[°]$	$\delta[\%]$
Šířka začátek trasy	90,35	0,35	0,39	89,98	0,02	0,02
Šířka uprostřed trasy	91,51	1,51	1,68	90,41	0,41	0,46
Šířka konec trasy	89,48	0,52	0,58	90,05	0,05	0,06
Délka	88,79	1,21	1,34	90,58	0,58	0,64
Aritmetický průměr					0,581	0,646

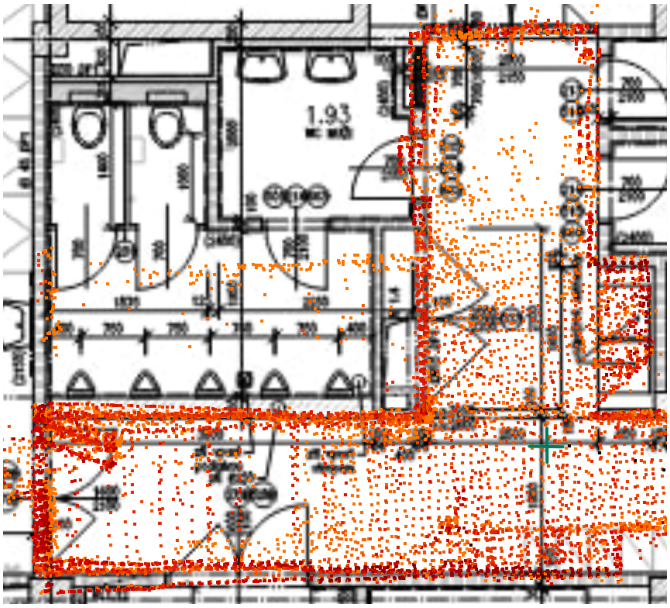
kde mapa vytváří bílé místo. U horní zdi místnosti se nachází stoly a dveře. Nad stoly jsou velká okna, přes které je vidět na protější budovu. I přesto, že jsou body snímané přes okna zašuměné, jde vidět na druhé straně koncentrace bodů mapy na stěně budovy.

Při podrobném porovnání dat z chodby a půdorysu lze vidět, že data nepůsobí jako přímka a vždy jsou úseky zdí, které směřují nerovnoběžně se zdí a po chvíli je nový nerovnoběžný úsek. Ve výsledku tyto úseky tvoří s drobnou odchylkou rovnou zeď. To může být způsobeno zpětnou optimalizací grafu, která upravuje získané pozice z odometrie, a tím zmenšuje chybu odometrie. Menší přesnosti a odchýlení od půdorysu se dostává částem, které jsou pozorovány pouze chvíli. Například při průjezdu kolem otevřených dveří místnosti jsou data rotovaná, případně nepatrně posunutá. Většinou data působí pouze lehce rotovaná, protože ve větších vzdálenostech se drobné otočení projeví více, než drobné posunutí. Při zaměření na detail v obrázku 6.2 je možné vidět, že se koridor mírně stáčí na pravou stranu. K natočení začíná docházet přibližně od poslední pozice. Při měření bylo výsledné natočení $3,69°$. Dá se předpokládat, že pokud by robot pokračoval v cestě dál, toto stočení by se pravděpodobně spravilo.



Obr. 6.2: Detail chybné rotace na konci prvního průjezdu s vyznačenou poslední pozicí

Při detailnějším prozkoumání druhého průjezdu, zvláště pak konce trasy na obrázku 6.3, je možné vidět také natočení. Tentokrát je větší než v minulých datech, ve spodní části dosahuje $8,83°$. Z mapy lze ovšem pozorovat, v případu této stěny, že četností převládají body se správnou polohou. K chybě pravděpodobně došlo nepřesným odhadem odometrie v určitém okamžiku. Při pokračování v cestě je možné, že optimalizace grafu by daná data mohla vylepšit díky většímu množství dat.

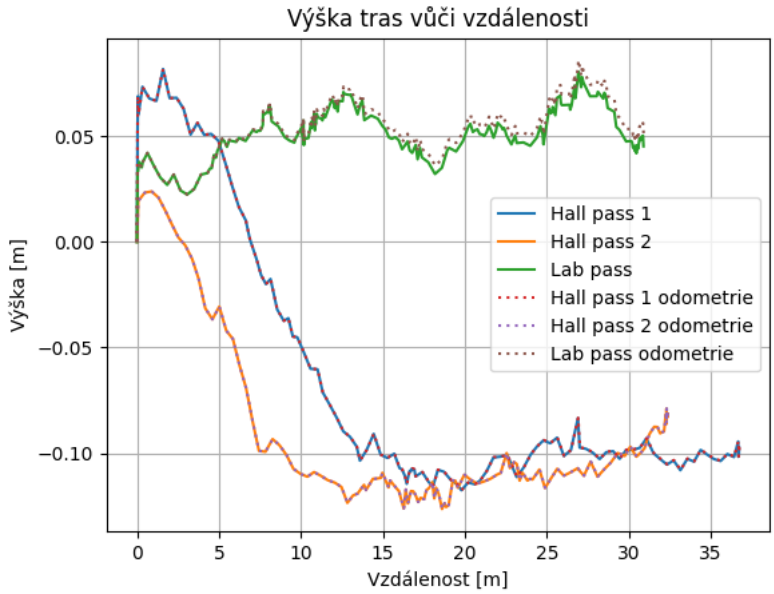


Obr. 6.3: Detail chybné rotace na konci druhého průjezdu s vyznačenou poslední pozicí

6.2.4 Průběh výšky

Jedna z věcí, s kterou má algoritmus nejviditelnější problém v rámci daných dat, je parametr výšky. Na obrázku 6.4 jsou zobrazeny průběhy odhadované výšky vůči ujeté vzdálenosti v souřadnicích x a y. Na průbězích z chodby je možné vidět, že algoritmus odhaduje pohyb směrem dolů. V druhé polovině se výška ustálí zhruba na stejné hodnotě, u dat z druhého průjezdu je možné pozorovat i drobný nárůst směrem vzhůru. Pokud by se brala pouze první část cesty vysvětlení by mohla být chyba způsobená v rámci daného robotického systému a snímače. Pravděpodobnějším vysvětlením je tendence algoritmu odhadovat směrem dolů, například pro nedostatek význačných bodů v okolí, ale proti tomu je například obrázek 4.3 z prvního testu s velkou velikostí filtru, kdy naopak dochází k nárůstu s mnohem méně daty. Druhá část dat se udržuje v podobné výšce, jedním z možných vysvětlení je optimalizace grafem, kdy existuje více dat, než na začátku trasy. Nicméně průběhy výšky

pro samotnou odometrii jsou skoro identické. Proto pravděpodobným důvodem je malé množství dat v globání mapě pro odometrii. Snímač obsahuje pouze 32 řádků. Robot proto musí chvíli jet, aby měl dostatek dat pro určení výšky na chodbě, která obsahuje vesměs mnoho velkých ploch. Místnost naopak obsahuje mnoho členitých předmětů a robot se tedy v rámci výšky snáze nachází, navíc je trasa následně optimalizována nalezením smyček, protože robot při pohybu udělá dvě kolečka uprostřed místnosti kolem stolu.



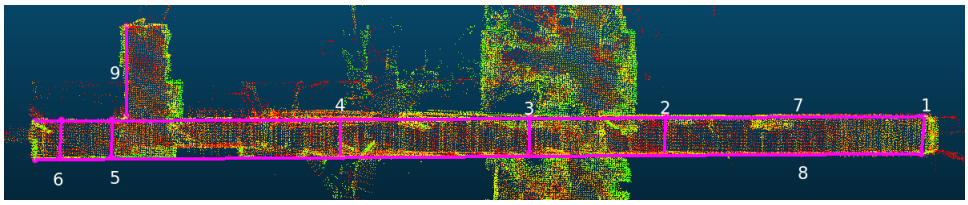
Obr. 6.4: Průběh odhadované výšky vůči ujeté vzdálenosti robota, tečkovaná data byla získána pouze na základě odometrie.

6.3 Online data

Při zkoušení na online datech bylo pořízeno několik kratších záznamů a jeden dlouhý. Dlouhý záznam projíždí chodbou a během toho vjede do dvou místností naproti sobě, s přejezdem mezi nimi. Algoritmus při přejezdu mezi místnosti měl větší chybu odhadu odometrie, jedním z důvodů je sloučení některých bodů z různých stran stěny mezi místností a chodbou. Jelikož stěna není o mnoho větší než velikost filtrů, může docházet i k sloučení některých bodů, což může vést k chybě velikosti stěny, která se následně projeví i na dalších datech zkosením. Po překonání tohoto místa pak algoritmus vrací hezkou mapu, nicméně již je posunuta oproti původnímu místu.

6.3.1 Měření vzdáleností

Místa, ve kterých byla měření provedena, jsou znázorněna na obrázku 6.5 a výsledky jsou uvedeny v tabulce 6.6. Měření je prováděno přibližné ve výšce půl metru. Číslování šířek chodby je z pravé strany na levou při pohledu na obrázek.



Obr. 6.5: Znázorněné měření ve výsledné mapě z online dat - řezy 1-6 jsou šířky s odpovídajícím číslováním, 7 - délka vpravo od robota, 8 - délka vlevo od robota, 9 - chodbička.

Přestože průměrná chyba vychází mírně horší než v případě offline dat, dala by se očekávat lepší data, protože trasa vede přes celou chodbu včetně chodbičky. Nejhorší data jsou z měření celkové délky chodby a třetí měřené šířky chodby, protože je značně ovlivněna nepřesným odhadem uprostřed mapy. V rámci měření ostatních vzdáleností si ale algoritmus zachovává podobnou přesnost jako při offline datech.

Tab. 6.6: Vzdálenosti z online dat

	x[m]	x_m[m]	Δ[m]	δ[%]
Šířka 1	1,8	1,791	0,009	0,50
Šířka 2	1,8	1,782	0,018	1,00
Šířka 3	1,8	1,701	0,099	5,50
Šířka 4	1,8	1,780	0,020	1,11
Šířka 5	1,8	1,795	0,005	0,28
Šířka 6	1,8	1,824	0,024	1,33
Délka vpravo do robota	43,2	42,776	0,424	0,98
Délka vlevo do robota	43,2	42,819	0,381	0,88
Chodbička	4,4	4,440	0,040	0,91
Aritmetický průměr			0,113	1,388

6.3.2 Měření úhlů

Měření kolmosti stěn pro chodbu bylo provedeno podobně jako offline měření. Měření bylo provedeno v naznačených řezech na obrázku 6.5. Výsledky měření se nachází

Tab. 6.7: Kolmost v online datech, φ_{m1} je úhel po pravé straně trasy nebo blíže startu, φ_{m2} je úhle po levé straně trasy nebo vzdálenější úhel od startu

	$\varphi_{m1}[°]$	$\Delta[°]$	$\delta[\%]$	$\varphi_{m2}[°]$	$\Delta[°]$	$\delta[\%]$
Šířka 1	88,35	1,65	1,83	91,85	1,85	2,05
Šířka 2	91,97	1,97	2,19	87,18	2,82	3,13
Šířka 3	90,91	0,91	1,01	87,88	2,12	2,36
Šířka 4	90,33	0,33	0,36	88,70	1,30	1,44
Šířka 5	91,59	1,59	1,77	89,68	0,32	0,35
Šířka 6	90,08	0,08	0,09	89,75	0,25	0,27
Délka vpravo do robota	93,12	3,12	3,47	86,74	3,26	3,62
Délka vlevo do robota	89,41	0,59	0,66	93,30	3,30	3,67
Chodbička	88,24	1,76	1.96	89,20	0,80	0,89
Aritmetický průměr					1,557	1,730

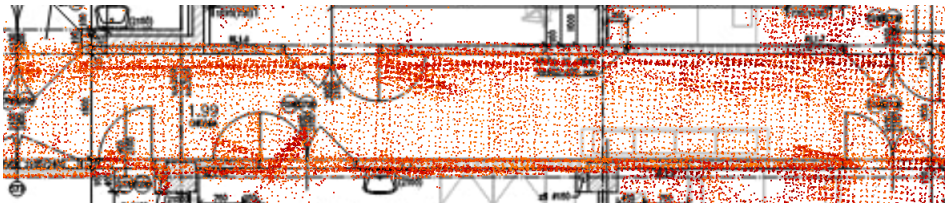
v tabulce 6.7, první údaj je měřen vždy buď po pravé straně robota, nebo nejblíže k počátku trasy, který se nachází napravo mapy. Měření úhlů již není tolik přesné jako měřené vzdáleností. Jedním z důvodů může být, že bodů snímaných nad úroveň robota je méně a často jsou snímány až z větší vzdálenosti.

6.3.3 Porovnání s půdorysem

Porovnání s půdorysem se nachází na obrázku B.6 v příloze. Při prozkoumání části chodby před vjezdem do místností a v místnostech samotných je mapa podobně přesná jako při offline datech. Vše souhlasí se svými hranicemi v půdorysu. Po výjezdu z první místnosti po levé straně dochází k chybě, která má za následek promítnutí dvou rozdílných poloh zdi do mapy. Problém s více hranicemi zmizí až v poslední části chodby, nicméně chodba je posunutá právě o rozdíl těchto dvou zdí. Na obrázku 6.6 je možné vidět dvě hranice stěny v podobném rozpětí, jako je šíře stěny. Pravděpodobný důvod je nerozlišení stěny z rozdílných stran po výjezdu z místnosti. Napomáhá tomu i skutečnost, že část stěny společná s místností je schovaná za otevřenými dveřmi a tyto body jsou nasnímány až po výjezdu z druhé místnosti, kdy se pokračuje v cestě chodbou.

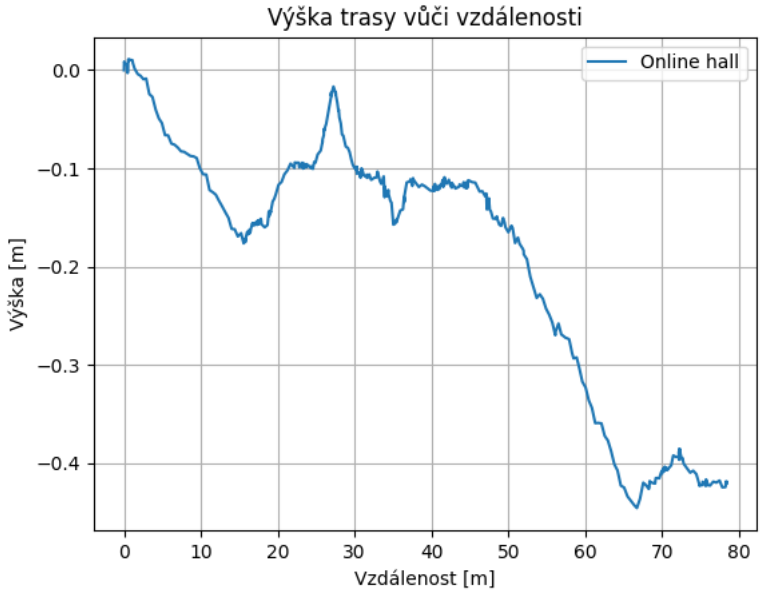
6.3.4 Průběh výšky

Průběh výšky se nachází na obrázku 6.7. V první části grafu je možné vidět klesání, u kterého se očekávalo, že se projeví, ze zkušenosti s offline daty. V momentě, kdy robot zajede do první místnosti, začne průběh naopak stoupat téměř až do původní



Obr. 6.6: Detail mapy online dat po výjezdu z místností

výšky. Protože robot couval z první místnosti ven po podobné trajektorii, jsou data na zpáteční cestě symetrická. Následný krátký vzestup výšky vzhůru reprezentuje přejezd přes chodbu do další místnosti. V následující části se robot nachází v druhé místnosti zde je držena přibližně konstantní výška. Při vjezdu zpět na chodbu robota doprovází strmý propad výšky dolů. Dle dat z offline měření by se v této části chodby mohlo předpokládat ustálení. Pravděpodobnou příčinou bude již zmiňované chybné zmapování stěny. Tedy robot o část dat, s kterými by bylo možné porovnat momentální sken, přichází nebo mohou zanášet větší chybu.



Obr. 6.7: Průběh výšky online dat

6.4 Zhodnocení

Při porovnání online a offline dat mnoho rozdílů nebylo. Při trase na chodbě byla největším problémem výška, příčinou je pravděpodobně málo bodů reprezentují-

cích hrany. Protože výška v místnostech, ač se pohybovala, tak neklesala. Klesání jako takové nemusí být nutně pravidlem jak je vidět na obrázku 4.3, kde jsou data ze snímače stejná a rozdílem je nastavení algoritmu, přesto výška naopak stoupá. Při porovnání online dat, kdy výška také občas stoupá, je možné dojít k závěru, že toto klesání a stoupání je způsobeno přenosem chyby, která je pouze změněna při výrazných změnách v prostředí či pohybu.

Dalším problém s kterým se algoritmus setkával byl členitý prostor. Při nastavení menší velikosti filtrů pro vstupní data dělal velmi členitý prostor problém, zvláště při náhlých změnách v prostředí, například při průjezdu v užším prostoru mezi stoly. Nastavením většího filtru je zlepšena robustnost, robot se orientuje více podle pevných bodů, jako jsou stěny, rohy a velké předměty. Velká velikost filtru snižuje přesnost, protože výsledný bod je aproximován ze všech dat snímače, tedy z bodů, které reprezentují malé předměty, kdy aproximací vynikají větší trendy, ale ty se blíží skutečnosti méně. Další zhoršení přesnosti při větším filtru je z důvodu menšího počtu bodů použitého pro odhad pozice. Nakonec velký filtr může narazit na podobný problém, který nastal v online mapě. Body objektu snímaného z více stran mohou být zaměněny za jeden bod. U menších předmětů to nemusí být nutně na škodu, ale pokud je velikost filtru větší než šířka zdi, kterou je v plánu snímat z obou stran, pravděpodobně budou data sjednocena a tím se zanese do mapy znatelná chyba.

Při výběr velikosti filtru pro tuto práci se bojovalo s oběma problémy, pro offline data z místnosti by menší velikost filtru nebyla ideální a algoritmus by s velkou chybou odhadl pozici a rotaci, výsledkem by bylo natočení celé místnosti. Pro registrování obou stran zdi by bylo možné zmenšit filtr aby se problému předešlo. Za nešťastné je možné pokládat i polohy dveří, které bránily zmapování dané stěny před příjezdem robota, a na danou oblast robot více viděl až po projetí obou místností. Ale to pouze vyloučilo možný scénář posunutí stěny v místnosti a umožnilo projevit se na chodbě.

Optimalizace grafem a hledání smyček často zpřesňovaly odometrii, ale pokud dojde v odometrii k větší chybě, tuto chybu již tato část algoritmu není schopna upravit. Jedním z možných problémů optimalizované mapy může být publikace jednou za 10 sekund, proto pro konec map, zvláště offline, kdy je algoritmus hned ukončen, mohlo dojít k tomu, že mapa nemusela být nutně aktualizována a proud dat se spolu s během algoritmu zastavil. Nicméně tento problém při je možné v při nasazení v budoucnu jednoduše odstranit, například publikováním a uložení výsledné mapy před koncem procesu.

Protože se snímač nacházel ve výšce půl metru nad zemí, a protože snímač má většinu paprsků nakloněnou směrem dolů, dosahuje mapa v mnoha místech pouze o něco větší výšky. Proto pro vygenerování plnohodnotného modelu místnosti nebo patra v budově, by pravděpodobně musel být snímač umístěn ve větší výšce aby

snímal i data ve větší výšce na bližší vzdálenost. Pro vytvoření základní mapy pro pohyb robota a lokalizace je tato výška dostačující.

Co se týče výpočetní náročnosti projektu, největší zatížení se zároveň na dvou jádrech procesoru pohybovalo okolo necelých 40%, zatížení ostatních jader se pohybovalo do 20%. Rychlost vypočítání odometrie se obvykle pohybuje okolo 25 ms při velkém množství bodů v okolí může trvat i dvojnásobně. Rychlost optimalizace grafu je značně závislá na velikosti mapy a počtu pozic, nicméně při největší mapě nepřesáhla 100 ms.

Přestože použitý projekt dosahuje velké přesnosti v proměnlivém vnitřním prostředí, chybí mu robustnost pro zvládnutí zároveň velmi členitého a málo členitého prostředí. Projekt je nejlepší upravit přímo pro danou lokaci a situace. Tato podmínka, ale nemůže být vždy splněna. Nicméně pro dosažení větší robustnosti je často nutné zmenšit nároky na přesnost. Pokud by se například dosáhlo větší robustnosti použitím větší velikosti filtrů na filtrování vstupních dat, může dojít k mnohem většímu zhoršení pro data málo členitých prostředí.

Použití algoritmu ve venkovním prostoru je určitě možné, důkazem je tomu prvotní použití na KITTI datasetu, který obsahuje data z LiDARu na střeše pomalu jedoucího auta. LiDAR snímá zároveň jak okolní členité prostředí i rovinu cesty, proto je možné, že algoritmus například při určování výšky může dosahovat lepších výsledků. I přesto by měl být vnitřní prostor jednodušší protože obsahuje velmi často předměty s definovanými tvary. Největší úskalí ve vnitřních prostorech bývá nedostatek objektů ve scéně. Pokud by algoritmus měl být použit ve venkovním prostředí na malém robotu, je otázkou, zda-li by došlo ke zhoršení. Pro použití s dronem může způsobit největší nevýhodu snímač, který je značně omezen ve snímání bodů v jedné souřadnici počtem řádků.

Závěr

Tato diplomová práce seznamuje se základní teorií a pojmy týkajícími se sebelokalizace a mapování, a především SLAMu s použitím 3D LiDARu. Následně shrnuje a porovnává momentálně nejmodernější open-source projekty pro řešení této úlohy. Z nich si následně vybírá projekt implementující optimalizovaný SC-F-LOAM, podrobně se mu věnuje a rozebere teorii za ním, a popíše základní architekturu jeho implementace. Před samotným oživením projektu je vytvořen návrh hardwaru a softwaru finálního systému. Po návrhu je systém zprovozněn na dodaných záznamech a poté je přenesen na mobilního robota. Nakonec jsou představeny a popsány výsledky.

U rešerše existujících projektů je možné povšimnout si, že existuje pouze několik principů, které se často vzájemně kombinují a upravují. Některé projekty tak často jsou jen vylepšení jiného projektu nebo kombinací dvou metod. Jedna z průlomových metod, která působí jako inspirace pro ostatní, je LOAM. Některé existující projekty se velmi často specificky zaměřují na vybrané podmínky, například velmi časté je zaměření na použití SLAMu pro autonomní vozidla, tím ale není omezeno jejich použití v jiných podmínkách. Úloha SLAMu ve venkovním prostředí je těžší a jsou obecně rozšířené datasety jak s více různými snímači, tak s daty s reálnou polohou. V některých projektech působí použití ve vnitřních prostorách opomenuté nebo je jen okrajově zmíněno. Jedním z problémů existujících řešení je velmi časté použití rozdílných vyhodnocení chyb, ale velké část projektů je vyhodnocena na KITTI datasetech, které umožňují jejich porovnání, minimálně pro použití ve vnějším prostředí.

Z popsaných projektů v první kapitole byl vybrán optimalizovaný SC-F-LOAM, který spojuje odometrii F-LOAM a k ní připojuje optimalizaci grafem a vyhledávání smyček pomocí deskriptoru ScanContext. Projekt patří k nejnovějším z popsaných a jako takový slibuje nejlepší výsledky v poměru výkon a přesnost. Navíc použití optimalizace grafu a hledání smyček může snížit velikost driftu v čase. Projekt je implementován v ROS1. Pro funkci algoritmu stačí na vstupu pouze sken z LiDARu, tedy mračno jedné otáčky snímače dokola. Projekt vrací několik hodnot v rámci ROS zpráv, výslednou mapu, upravenou odometrii a trasu. Navíc projekt ukládá výslednou mapu a zaznamenává pozice do souboru.

V třetí kapitole byly představeny hardwarové možnosti, které jsou dostupné. Dále byly popsány LiDAR snímače, jejich principy a úskalí při jejich výběru. Po popsání hardwarové části byl vytvořen návrh softwarového systému, popsány důvody proč byl zvolen pro zprovoznění projektu Docker a z toho plynoucí výhody a nevýhody.

Čtvrtá a pátá kapitola se věnují samotné implementaci a oživování projektu na offline a online záznamech. V kapitolách jsou popsána úskalí, se kterými bylo

nutné se potýkat při oživování, a jakým způsobem byl projekt laděn na existující podmínky. Kapitoly nepředkládají a neukazují výsledky, ty jsou popsány v poslední části práce.

Poslední část práce definuje způsoby vyhodnocení a následně jsou výsledky z předchozích dvou kapitol posuzovány na jejich základě. Rozdíl mezi výsledky offline dat a online dat je minimální a většinou souvisí se specifickými situacemi.

Z offline dat je možné dojít k závěru, že pro přesná data je třeba mít pohled na tato data z více pozic. Při okrajových datech, například na konci záznamu, dochází k větší chybě. Rozměry pro filtrování dat byly odvozeny v offline datech, a pro menší rozměry se algoritmus v jednom z datasetů ztrácel. Dále se zde ukázalo špatné určení výšky robota, které se při cestě chodbou projevovalo zvláště ze začátku. Pravděpodobným důvodem je nedostatečně rozmanité prostředí pro určení výšky v dané mapě. Později, kdy je zmapována větší část chodby, se výška více ustálí.

Při pořizování online dat byl objeven problém s filtrováním dat, který ukazovaná data velmi ovlivnil. Ten byl způsoben kombinací dat ze dvou stran stěny. Řešením tohoto problému by bylo zmenšení filtru, ale pak by se ztratilo na robustnosti, to by zapříčinilo ztrátu robota v jednom z offline záznamů a výsledek z něj by nebyl použitelný. Naproti tomu chyba u online dat pouze způsobila pokles v kvalitě výsledných dat a algoritmus byl schopen danou část vyřešit.

Při testování online byla vypozorována špatná reakce algoritmu na prudké, rychlé a trhavé pohyby. To je ale vlastnost, s kterou se bude potýká většina momentálně existujících řešení a pouze některé z popsaných se snaží tuto problematiku více řešit, protože se nejvíce týká menších mobilních robotů pohybujících v terénu.

Jedna z možností, jak by se dalo řešit omezení 3D LiDARu ve snímání výšky a tím i větší prostor pro zlepšení v rámci vyhodnocení, je použití více snímačů. Pravděpodobně by bylo možné přepracovat vybraný projekt pro práci s více snímači, ale rozhodně by to vyžadovalo velký zásah do kódu a není zaručena, že výsledek mnoho zlepšení nepřinese. Protože existují projekty, které jsou vytvořeny přímo pro více snímačů nebo které se snaží pro větší univerzálnost a podporují jeden a více rozdílných snímačů, lepší řešením by bylo zvolit jiný projekt, který by splňoval podmínku použití s více LiDARy.

Vybranou metodu hodnotím jako přesnou, nikoliv jako univerzální. Je potřeba na základě znalosti prostředí algoritmus upravit, poté může vracet velmi dobré výsledky. Při výběru existujících řešení tohoto problému pro reálnou aplikaci bych doporučil brát v potaz i použití dané metody ve více projektech, kdy nejenže jsou vyladěny drobné chyby, ale je více zřejmé, co od daného algoritmu očekávat a jaká prostředí jsou pro něj vhodná či naopak. Použití daného algoritmu ve vnější prostředí je možné. Nic nenasvědčuje tomu, že by algoritmus měl mít znatelně horší přesnost, ač pravděpodobně k mírnému zhoršení přesnosti dojde.

Literatura

[1] CADENA, Cesar, Luca CARLONE, Henry CARRILLO, Yasir LATIF, Davide SCARAMUZZA, José NEIRA, Ian REID a John J LEONARD. Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age: Toward the Robust-Perception Age. *IEEE Transactions on Robotics*. 2016, **32**(6), 1309-1332. Dostupné z: doi:10.1109/TRO.2016.2624754

[2] RUSINKIEWICZ, S a M LEVOY. *Efficient variants of the ICP algorithm*. 2001, 145-152. Dostupné z: doi:10.1109/IM.2001.924423

[3] ZHANG, Ji a Sanjiv SINGH. Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*. 2017, **41**(2), 401-416.

[4] LABOSHINL *Loam_velodyne* [online]. [cit. 2022-12-12]. Dostupné z: https://github.com/laboshinl/loam_velodyne

[5] HKUST-AERIAL-ROBOTICS *A-LOAM* [online]. [cit. 2022-12-12]. Dostupné z: https://github.com/HKUST-Aerial-Robotics/A-LOAM

[6] GISBI-KIM *SC-A-LOAM* [online]. [cit. 2022-12-14]. Dostupné z: https://github.com/gisbi-kim/SC-A-LOAM

[7] SHAN, Tixiao a Brendan ENGLOT. *LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain*. 2018, 4758-4765. Dostupné z: doi:10.1109/IROS.2018.8594299

[8] ROBUSTFIELDAUTONOMYLAB *LeGO-LOAM* [online]. [cit. 2022-12-14]. Dostupné z: https://github.com/RobustFieldAutonomyLab/LeGO-LOAM

[9] IRAPKAIST *SC-LeGO-LOAM* [online]. [cit. 2022-12-14]. Dostupné z: https://github.com/irapkaist/SC-LeGO-LOAM

[10] WANG, Han, Chen WANG, Chun-Lin CHEN a Lihua XIE. *F-LOAM: Fast LiDAR Odometry and Mapping: Fast LiDAR Odometry and Mapping*. 2021, 4390-4396. Dostupné z: doi:10.1109/IROS51168.2021.9636655

[11] LIAO, Lizhou, Chunyun FU, Binbin FENG a Tian SU. *Optimized SC-F-LOAM: Optimized Fast LiDAR Odometry and Mapping Using Scan Context: Optimized Fast LiDAR Odometry and Mapping Using Scan Context*. 2022, 1-6. Dostupné z: doi:10.1109/CVCI56766.2022.9964574

[12] KOIDE, Kenji, Jun MIURA a Emanuele MENEGATTI. A portable three-dimensional LIDAR-based system for long-term and wide-area people behavior measurement. *International journal of advanced robotic systems*. London, England: SAGE Publications, 2019, **16**(2), 172988141984153-172988141981694. ISSN 1729-8806. Dostupné z: doi:10.1177/1729881419841532

[13] KOIDE3 *Hdl_graph_slam* [online]. [cit. 2022-12-15]. Dostupné z: https://github.com/koide3/hdl_graph_slam

[14] FROSI, Matteo a Matteo MATTEUCCI. ART-SLAM: Accurate Real-Time 6DoF LiDAR SLAM: Accurate Real-Time 6DoF LiDAR SLAM. *IEEE Robotics and Automation Letters*. 2022, **7**(2), 2692-2699. Dostupné z: doi:10.1109/LRA.2022.3144795

[15] BEHLEY, Jens a Cyrill STACHNISS. Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments. *Robotics: Science and Systems XIV*. 2018. Dostupné z: doi:10.15607/RSS.2018.XIV.016

[16] CHEN, Xieyuanli, Andres MILIOTO, Emanuele PALAZZOLO, Philippe GIGUÈRE, Jens BEHLEY a Cyrill STACHNISS. *SuMa++: Efficient LiDAR-based Semantic SLAM: Efficient LiDAR-based Semantic SLAM*. 2019, 4530-4537. Dostupné z: doi:10.1109/IROS40897.2019.8967704

[17] LIN, Jiarong a Fu ZHANG. Loam_livox: A fast, robust, high-precision LiDAR odometry and mapping package for LiDARs of small FoV: A fast, robust, high-precision LiDAR odometry and mapping package for LiDARs of small FoV. *ArXiv. org*. 2019.

[18] WANG, Han, Chen WANG a Lihua XIE. Lightweight 3-D Localization and Mapping for Solid-State LiDAR. *IEEE Robotics and Automation Letters*. 2021, **6**(2), 1801-1807. Dostupné z: doi:10.1109/LRA.2021.3060392

[19] GEIGER, Andreas, Philip LENZ a Raquel URTASUN. *Are we ready for autonomous driving? The KITTI vision benchmark suite*. 2012, 3354-3361. Dostupné z: doi:10.1109/CVPR.2012.6248074

[20] KIM, Giseop a Ayoung KIM. *Scan Context: Egocentric Spatial Descriptor for Place Recognition Within 3D Point Cloud Map: Egocentric Spatial Descriptor for Place Recognition Within 3D Point Cloud Map*. 2018, 4802-4809. Dostupné z: doi:10.1109/IROS.2018.8593953

[21] *ROS Wiki* [online]. [cit. 2023-04-04]. Dostupné z: https://wiki.ros.org/

[22] ELECTRONICS, Digi-Key. *Simplifying Time-of-Flight Distance Measurements* [online]. 2017 [cit. 2023-01-04]. Dostupné z: https://www.digikey.dk/da/articles/simplifying-time-of-flight-distance-measurements

[23] INTEL® *Intel® RealSense LiDAR Camera L515* [online]. [cit. 2023-01-04]. Dostupné z: https://www.intelrealsense.com/lidar-camera-l515/

[24] LIVOX *LIVOX Avia* [online]. [cit. 2023-01-04]. Dostupné z: https://www.livoxtech.com/avia/specs

[25] LIVOX *LIVOX Mid-40/Mid-100* [online]. [cit. 2023-01-04]. Dostupné z: https://www.livoxtech.com/mid-40-and-mid-100

[26] LIVOX *LIVOX Mid-70* [online]. [cit. 2023-01-04]. Dostupné z: https://www.livoxtech.com/mid-70

[27] LAMBERT, Jacob, Alexander CARBALLO, Abraham MONRROY, Patiphon NARKSRI, David WONG, Eijiro TAKEUCHI a Kazuya TAKEDA. Performance Analysis of 10 Models of 3D LiDARs for Automated Driving. *IEEE Access*. 2020, **PP**, 1-1. Dostupné z: doi:10.1109/ACCESS.2020.3009680

[28] *Docker* [online]. [cit. 2023-05-04]. Dostupné z: https://www.docker.com/

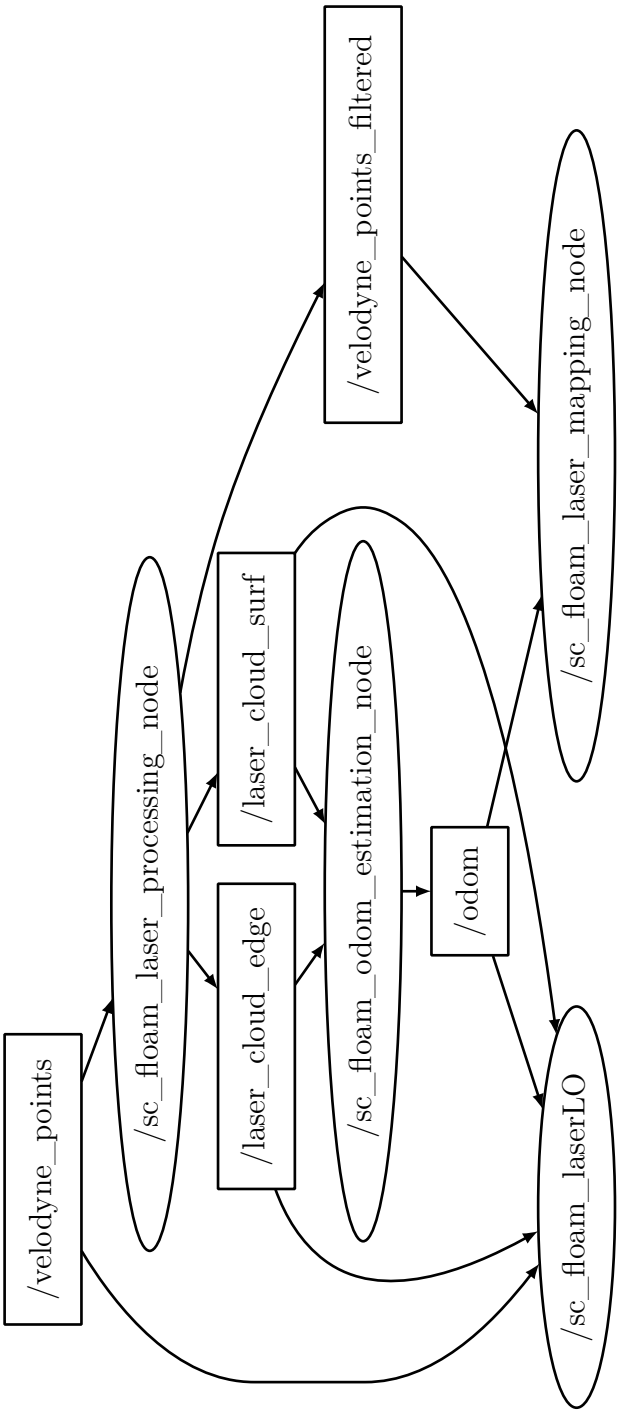
[29] *OSRF Docker Images* [online]. [cit. 2023-05-04]. Dostupné z: https://github.com/osrf/docker_images

[30] *Using GUIs with Docker* [online]. [cit. 2023-03-11]. Dostupné z: https://wiki.ros.org/docker/Tutorials/GUI

[31] *Rosbags* [online]. [cit. 2023-05-04]. Dostupné z: https://gitlab.com/ternaris/rosbags

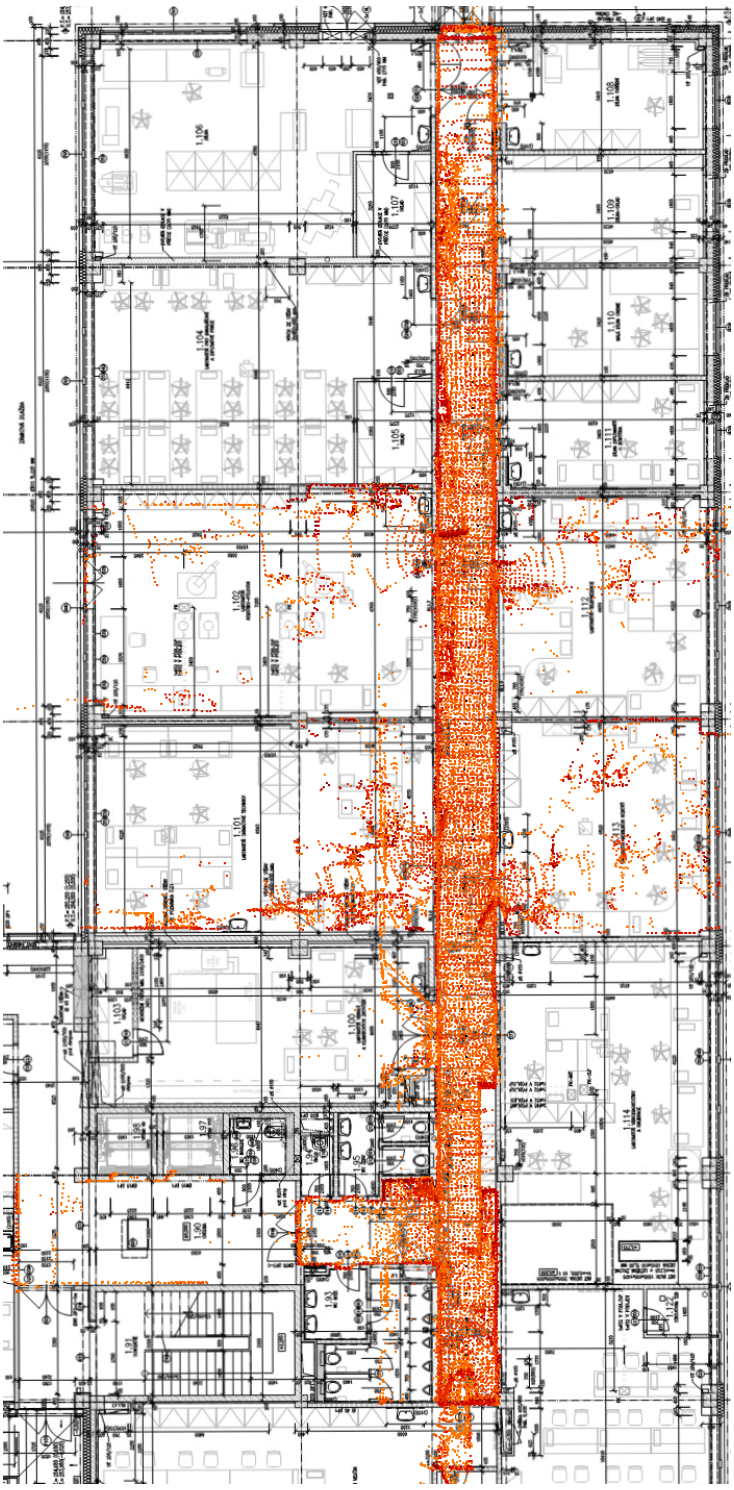
[32] *Downsampling a PointCloud using a VoxelGrid filter* [online]. [cit. 2023-04-13]. Dostupné z: https://pcl.readthedocs.io/projects/tutorials/en/latest/voxel_grid.html

A Schema implementace

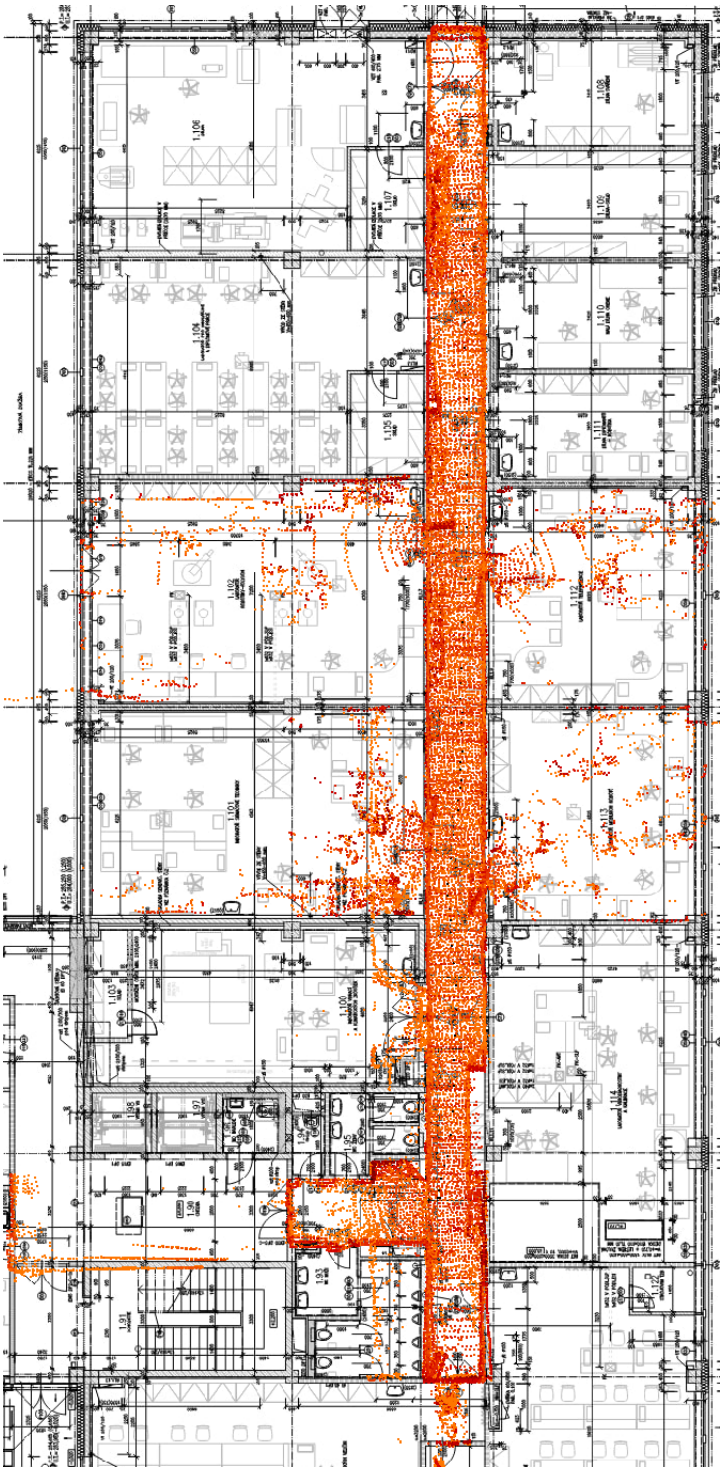


Obr. A.1: Graf implementace optimized SC-FLOAM v ROSu, v obdélnících jsou odebírané topic a v elipsách node

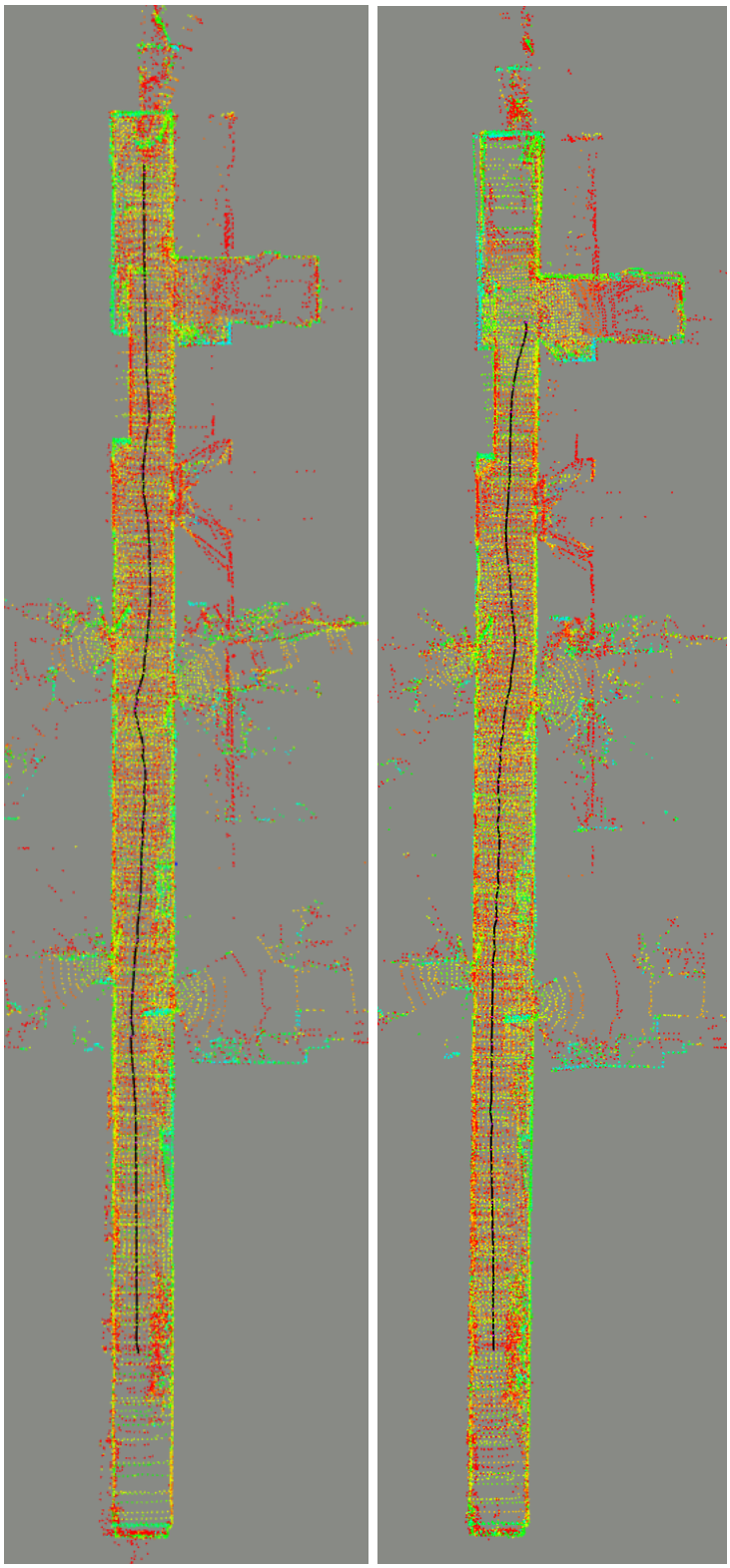
B Mapy



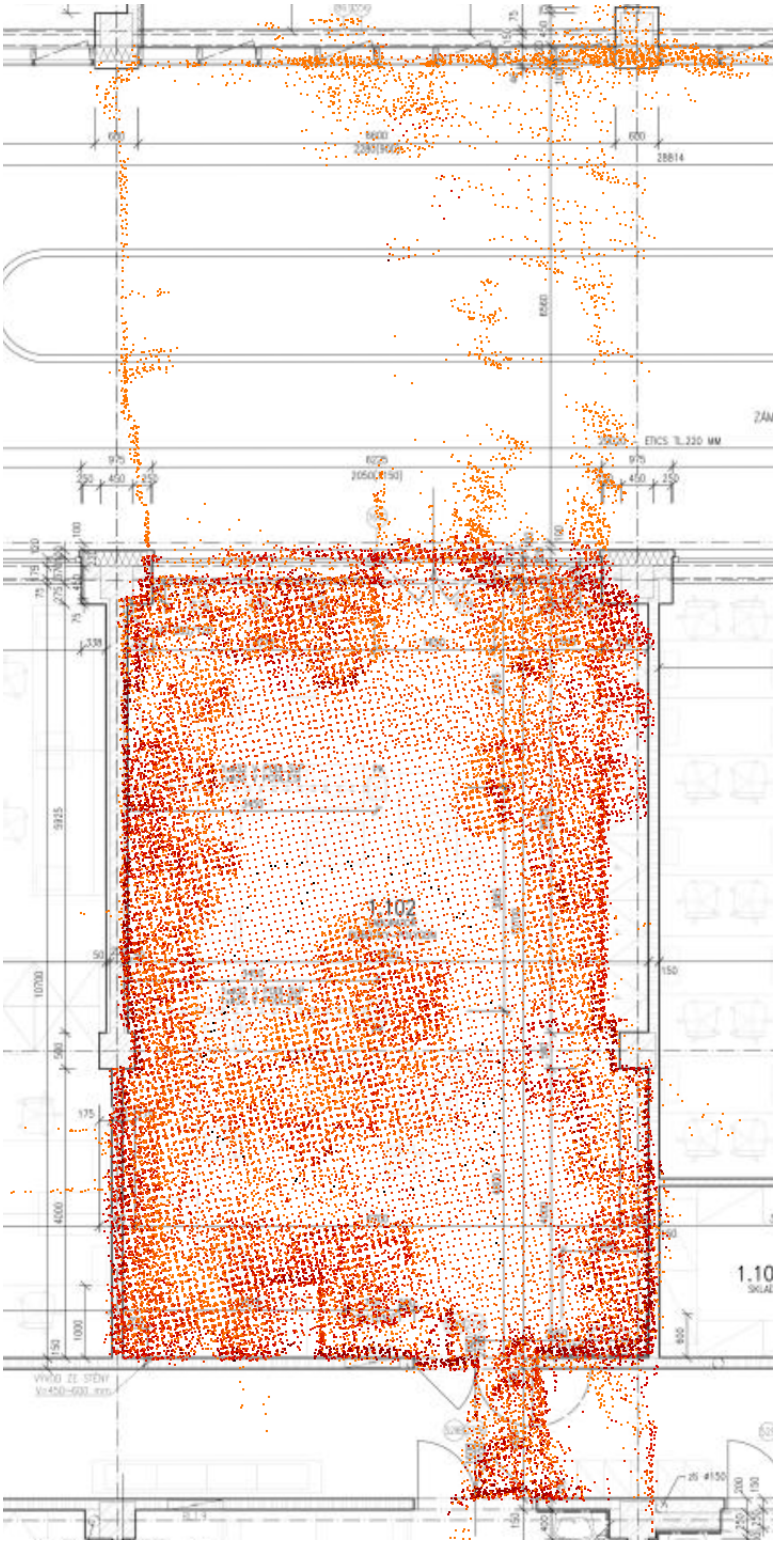
Obr. B.1: Porovnání výsledné mapy s plánem budovy, na obrázku jsou výsledky dat *hall_pass_1*



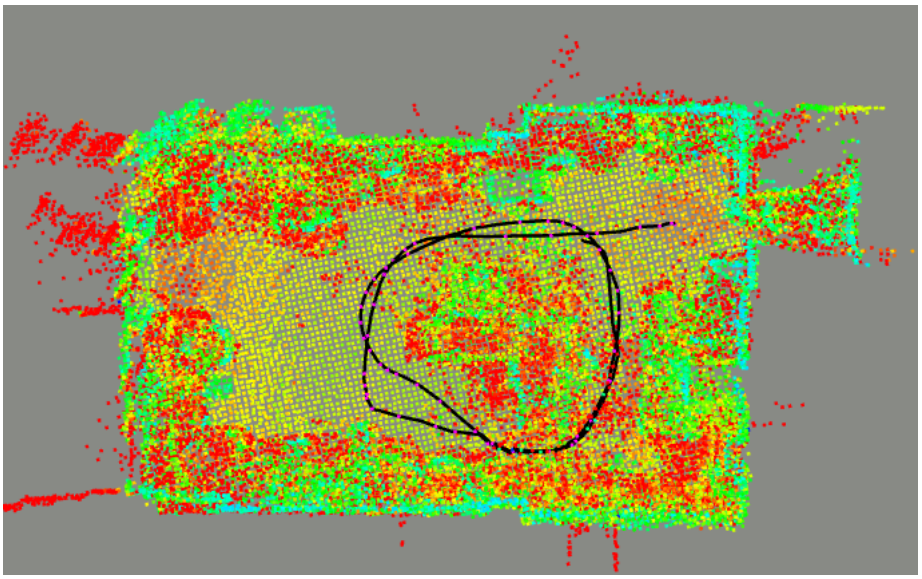
Obr. B.2: Porovnání výsledné mapy s plánem budovy, na obrázku jsou výsledky dat *hall_pass_2*



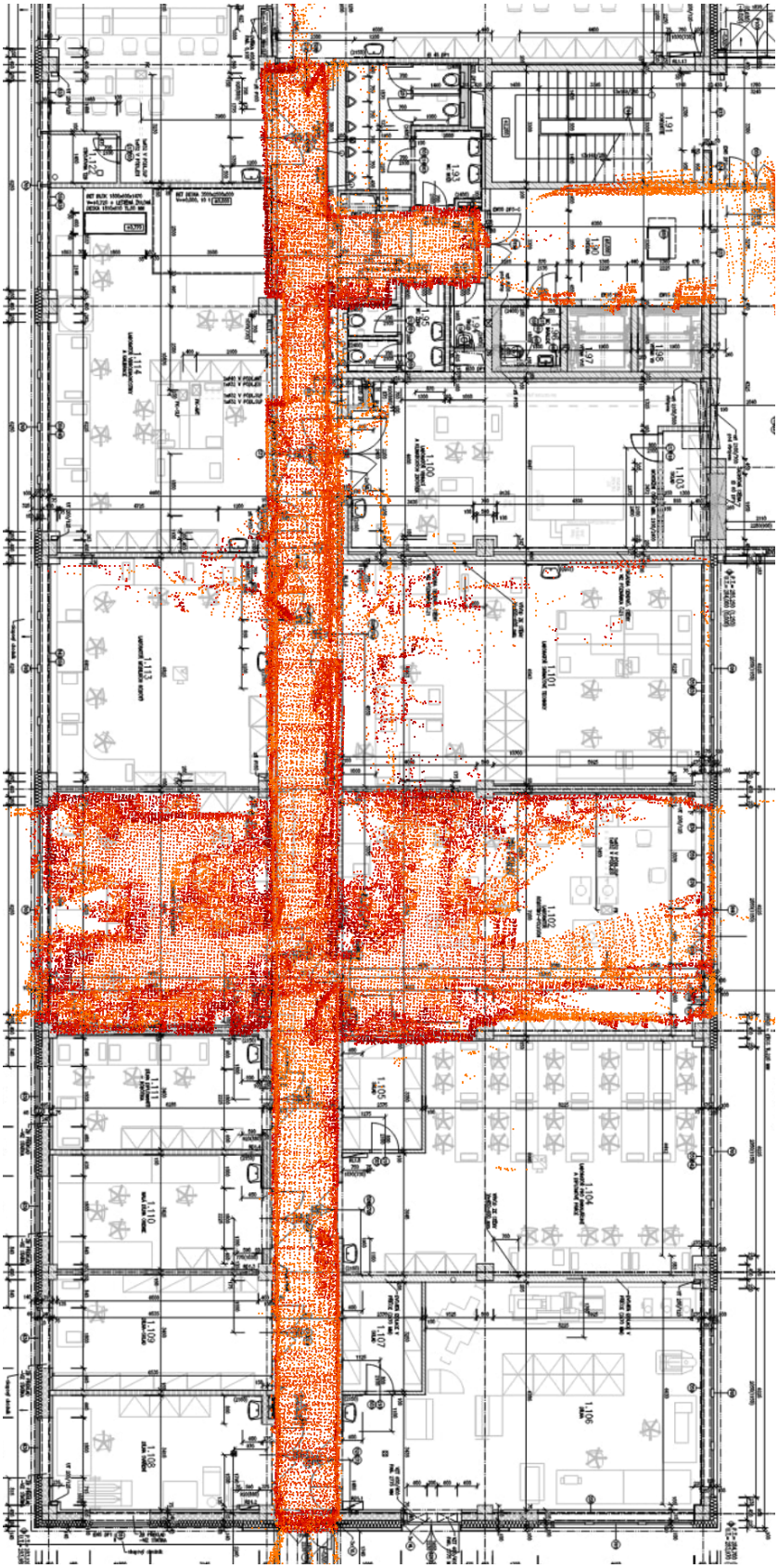
Obr. B.3: Mapy chodby s trasou na levé straně je záznam z *hall_pass_1*, start trasy je v horní části, na pravé straně je záznam z *hall_pass_2* se startem ve spodní části



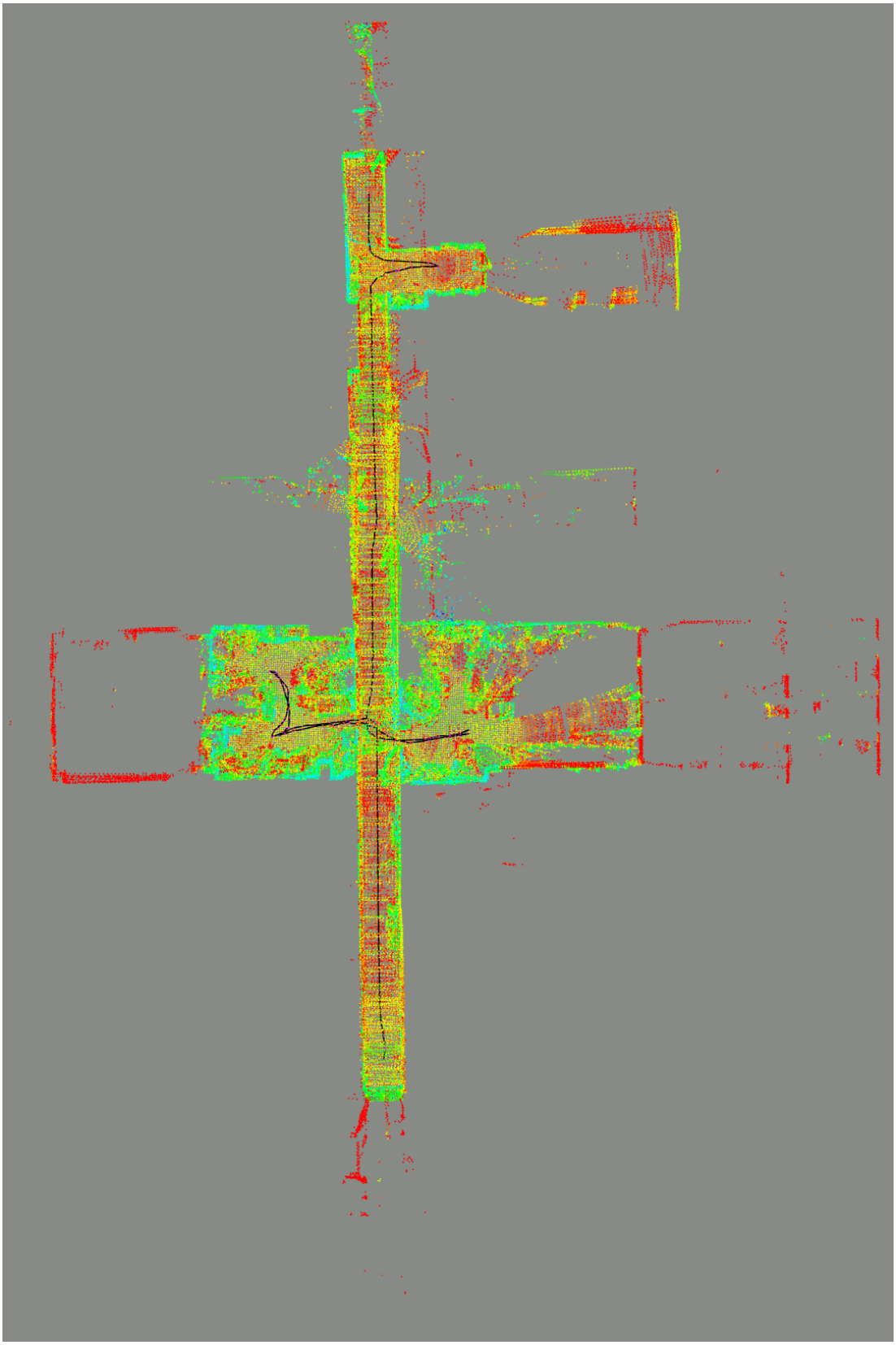
Obr. B.4: Porovnání výsledné mapy s plánem budovy, na obrázku jsou výsledky dat *lab_pass_1*



Obr. B.5: Mapa místnosti s trasou ze záznamu *lab_pass_1*, start trasy se nachází v pravé horní části



Obr. B.6: Porovnání mapy online dat a půdorysu budovy



Obr. B.7: Mapy online dat s trasou, začátek trasy se nachází ve spodní části obrázku

C Obsah elektronické přílohy

V informačním systému bude uložena celá elektronická příloha, kromě složky *bags* a zdrojového kódu textu práce, která obsahuje nahrané záznamy v bag souborech. Ve složce *latex* je zdrojový kód práce a ve složce *Optimized-SC-F-LOAM* je kód vybraného projektu včetně úprav udělaných během této práce. Následující strom souborů je pro svoji velikost rozdělen na několik stran, pro lepší přehlednost strom vždy začíná nejvyšším adresářem:

```
Diplomova_prace
├── thesis.pdf
└── Optimized-SC-F-LOAM
    ├── .DS_Store
    ├── LICENSE
    ├── README.md
    ├── docker
    │   ├── .bashrc
    │   ├── Dockerfile
    │   ├── bridge.yaml
    │   ├── run_bridge.sh
    │   ├── run_docker.sh
    │   └── run_ros2_docker.sh
    └── optimized-sc-f-loam
        ├── .DS_Store
        ├── CMakeLists.txt
        ├── package.xml
        ├── include
        │   ├── laserLoopOptimizationClass.h
        │   ├── laserMappingClass.h
        │   ├── laserProcessingClass.h
        │   ├── lidar.h
        │   ├── lidarOptimization.h
        │   ├── odomEstimationClass.h
        │   └── scancontext
        │       ├── KDTreeVectorOfVectorsAdaptor.h
        │       ├── Scancontext.cpp
        │       ├── Scancontext.h
        │       ├── common.h
        │       ├── nanoflann.hpp
        │       └── tic_toc.h
        └── launch
            ├── .DS_Store
            ├── optimized_sc_f_loam_mapping.launch
            ├── thesis_offline.launch
            └── thesis_online.launch
```

```
Diplomova_prace
├── Optimized-SC-F-LOAM
│   └── optimized-sc-f-loam
│       ├── rviz
│       │   ├── .DS_Store
│       │   └── optimized_sc_floam_mapping.rviz
│       └── src
│           ├── laserLoopOptimizationClass.cpp
│           ├── laserLoopOptimizationNode.cpp
│           ├── laserMappingClass.cpp
│           ├── laserMappingNode.cpp
│           ├── laserProcessingClass.cpp
│           ├── laserProcessingNode.cpp
│           ├── lidar.cpp
│           ├── lidarOptimization.cpp
│           ├── odomEstimationClass.cpp
│           └── odomEstimationNode.cpp
├── bags
│   ├── hall_online.bag
│   ├── hall_pass_1
│   │   ├── hall_pass_1_0.db3
│   │   └── metadata.yaml
│   ├── hall_pass_2
│   │   ├── hall_pass_2_0.db3
│   │   └── metadata.yaml
│   └── lab_pass_1
│       ├── lab_pass_1_0.db3
│       └── metadata.yaml
└── latex
    ├── T1-Vafle-Regular-base.tfm
    ├── T1-Vafle-Regular-lcdfj.tfm
    ├── T1-Vafle-Regular.tfm
    ├── T1-Vafle-Regular.vf
    ├── VafleVUT-Regular.pfb
    ├── VafleVUT-RegularLCDFJ.pfb
    ├── a_trqo3o.enc
    ├── nastaveni.tex
    ├── sablona-prace.tex
    ├── t1vafle.fd
    ├── thesis.sty
    └── vafle.map
```

```
Diplomova_prace
└── latex
    └── obrazky
        ├── 1_metody
        │   ├── Suma_schema.png
        │   ├── art_slam_modules.png
        │   ├── factor_graph.png
        │   └── lego_loam_ground.png
        ├── 2_metoda
        │   ├── ScanContextFirst.png
        │   ├── ScanContextSecond.png
        │   └── schema.png
        ├── 3_navrh
        │   ├── ToF.png
        │   └── solid-state-lidar.png
        ├── 4_offline
        │   ├── hall1first.png
        │   ├── hallfirstvyska.png
        │   ├── odometry.png
        │   └── roz_odom.png
        ├── 6_vysledky
        │   ├── offline
        │   │   ├── hall1_plan.png
        │   │   ├── hall1_pudorys_detail.png
        │   │   ├── hall1_vzdalenosti.png
        │   │   ├── hall2_plan.png
        │   │   ├── hall2_pudorys_detail.png
        │   │   ├── hall2_vzdalenosti.png
        │   │   └── offline_vyska.png
        │   └── online
        │       ├── hall_detail.png
        │       ├── hall_mereni.png
        │       └── online_vyska.png
        ├── prilohy
        │   ├── hall1_pudorys.png
        │   ├── hall1_trasa.png
        │   ├── hall2_pudorys.png
        │   ├── hall2_trasa.png
        │   ├── hall_pudorys.png
        │   ├── hall_trasa.png
        │   ├── lab_pudorys.png
        │   └── lab_trasa.png
        └── schemata
            ├── node.tikz
            └── tf_dataset.tikz
```

```
Diplomova_prace
├── latex
│   ├── pdf
│   │   ├── student-titulka.pdf
│   │   └── student-zadani.pdf
│   └── text
│       ├── 1_metody.tex
│       ├── 2_metoda.tex
│       ├── 3_navrh.tex
│       ├── 4_offline.tex
│       ├── 5_online.tex
│       ├── 6_vysledky.tex
│       ├── literatura.bib
│       ├── literatura.tex
│       ├── prilohy.tex
│       ├── uvod.tex
│       └── zaver.tex
└── results
    ├── hall1
    │   ├── gt_poses.txt
    │   ├── odom_poses.txt
    │   ├── optimized_poses.txt
    │   └── Scans
    │       └── map.pcd
    ├── hall2
    │   ├── gt_poses.txt
    │   ├── odom_poses.txt
    │   ├── optimized_poses.txt
    │   └── Scans
    │       └── map.pcd
    ├── lab1
    │   ├── gt_poses.txt
    │   ├── odom_poses.txt
    │   ├── optimized_poses.txt
    │   └── Scans
    │       └── map.pcd
    └── online_hall
        ├── odom_poses.txt
        ├── optimized_poses.txt
        └── Scans
            └── map.pcd
```