
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV INTELIGENTNÍCH SYSTÉMŮ**
**DEPARTMENT OF INTELLIGENT SYSTEMS**


# LIDAR A STEREOKAMERA V LOKALIZACI MOBILNÍCH ROBOTŮ

**LIDAR AND STEREOCAMERA IN LOCALIZATION OF MOBILE ROBOTS**

**DIPLOMOVÁ PRÁCE**
**MASTER'S THESIS**


**AUTOR PRÁCE** — **Bc. JANA VYROUBALOVÁ**
**AUTHOR**

**VEDOUCÍ PRÁCE** — **Ing. FILIP ORSÁG, Ph.D.**
**SUPERVISOR**



**BRNO 2017**

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav inteligentních systémů

Akademický rok 2016/2017

# Zadání diplomové práce

Řešitel: **Vyroubalová Jana, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **LIDAR a stereokamera v lokalizaci mobilních robotů**
**LIDAR and Stereocamera in Localization of Mobile Robots**
Kategorie: Umělá inteligence

Pokyny:

1. Prostudujte problematiku lokalizace mobilních robotů a mapování prostoru. Zaměřte se na tvorbu map z prostorových dat získaných LIDARem a stereokamerou.
2. Seznamte se s mobilním robotem RUDA, systémem ROS (Robotický Operační Systém) a způsobem integrace a využití dat z LIDARu a stereokamery pro lokalizaci robota RUDA.
3. Navrhněte softwarový modul lokalizace pro ROS integrující data z LIDARu a stereokamery, navržený modul implementujte.
4. Proveďte základní experimenty s robotem RUDA a implementovaným modulem. Vyhodnoťte kvalitu lokalizace vzhledem k parametrům robota.
5. Shrňte dosažené výsledky a navrhněte další možnosti vývoje.

Literatura:

- FERNANDEZ-MADRIGAL, Juan-Antonio, BLANCO CLARACO, Jose Luis. *Simultaneous localization and mapping for mobile robots: introduction and methods*. Hershey, PA: Information Science Reference, 2013. ISBN 9781466621060

Při obhajobě semestrální části projektu je požadováno:

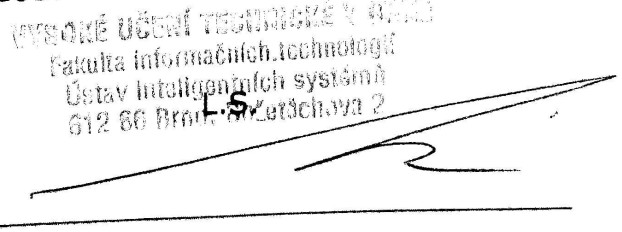
- Body 1 a 2.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Orság Filip, Ing., Ph.D.**, UITS FIT VUT
Konzultant: Luža Radim, Ing., UITS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017




doc. Dr. Ing. Petr Hanáček
*vedoucí ústavu*

# Abstrakt

Lidar (2D) se velmi často používá v mapování, lokalizaci a navigaci mobilních robotů. Jeho využití má však svá omezení a to převším využitelnost spíše v jednoduchém prostředí. Tento problém může být odstraněn přidáním dalších senzorů a jejich zpracování dat. Naše práce představuje metodu, jak spolu mohou být data ze stereo kamery a LIDARu fúzována za účelem lepšího dynamického mapování. Za prerekvizitu považujeme 2D mapu obsazenosti z LIDARu, která je rozšířena 2D mapou opsazenosti získanou ze stereo kamery. Princip našeho přístupu je založen na detekci pozemní roviny v disparitní mapě získané ze stereo vize. Pro detekci pozemní roviny využíváme metod RANSAC a Metody nejmenších čtverců. Dále po určení překážek v disparitní mapě generujeme z těchto informací 2D mapu obsazenosti. Výstupem naší metody je 2D mapa, která je tedy výsledkem fúze doplňujících se informací ze senzorů LIDARu a stereo kamery. Experimentální výsledky získány z dat školního robota RUDA a data setu MIT Stata Center Data Set jsou dostatečné na to, abychom mohli prohlásit tuto metodu za velký přínos, ačkoliv je naše implementace pouze prototypem. Kromě prezentace našeho přístupu zde také analyzujeme výstupy a diskutujeme různá vylepšení a rozšíření za získáním lepších výsledků.



# Abstract

LIDAR (2D) has been widely used for mapping, localization and navigation in mobile robotics. However, its usage is limited to simple environments. This problem can be solved by adding more sensors and processing these data together. This paper explores a method how measurements from a stereo camera and LIDAR are fused to dynamical mapping. An occupancy grid map from LIDAR data is used as prerequisite and extended by a 2D grid map from stereo camera. This approach is based on the ground plane estimation in disparity map acquired from the stereo vision. For the ground plane detection, RANSAC and Least Squares methods are used. After obstacles determination, 2D occupancy map is generated. The output of this method is 2D map as a fusion of complementary maps from LIDAR and camera. Experimental results obtained from RUDA robot and MIT Stata Center Data Set are good enough to determine that this method is a benefit, although my implementation is still a prototype. In this paper, we present the applied methods, analyze the results and discuss the modifications and possible extensions to get better results.



# Klíčová slova

LIDAR, stereo kamera, mapování, Metoda nejmenších čtverců ve 3D, fúze senzorů, disparitní mapa, detekce pozemní roviny, mobilní robotika

# Keywords

LIDAR, stereo camera, mapping, Least square plane method, sensors fusion, disparity map, ground plane detection, mobile robotics

# Citace

VYROUBALOVÁ, Jana. *LIDAR a stereokamera v lokalizaci mobilních robotů*. Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Filip Orság, Ph.D.

# LIDAR a stereokamera v lokalizaci mobilních robotů

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně pod vedením Ing. Filipa orsága, Ph.D. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

.....
Jana Vyroubalová
21. května 2017

## Poděkování

Děkuji svému vedoucímu za pohodové a vyhovující vedení práce. Dále děkuji Radimu Lužovi za poskytování odborných konzultací. A v neposlední řadě musím poděkovat svému králíkovi (Letyna) za to, že mě udržoval vzhůru při čtení odborných článků.

# Obsah

# Kapitola 1

# Úvod

V dnešní době je robotika všude kolem nás. Lidé se snaží více a více automatizovat (například v odvětví průmyslu a domácnostech). V automobilové dopravě se kolem nás již běžně pohybují vozidla s asistenčnímy systémy a velmi vyjímečně už se dostávají do provozu i autonomní[1] vozidla. Ve zdravotnictví jsou dnes lékaři díky moderní robotice schopni provádět zákroky, na které již pouhé lidské síly nestačí.

Robotika se stále vyvíjí a řekla bych, že je to v posledních letech právě velmi moderní téma, které zažívá obrovský rozvoj a to jak na straně hardwaru, tak na straně softwaru. A právě v této práci se budu zabývat řešením problému, ve kterém se nachází snad každý mobilní robot. Jedná se o fakt, že takový robot potřebuje znát svoji pozici vůči okolí a zároveň potřebuje mít právě toto lokální okolí nějakým způsobem zmapované. Pro primitivní roboty to samozřejmě není nutnost, ale zde mluvíme o užitečných mobilních robotech. Taková mapa se dá potom použít například pro další krok v robotice a tím je plánování trasy robota. Tedy pokud chceme aby robot někam dojel, musíme mu říct kudy má jet, což by se neobešlo bez znalosti okolí.

Tvorba mapy a lokalizace robota v ní prochází několika problémy. K tvorbě mapy se využívájí vhodné senzory, které ale samozřejmě nejsou dokonalé. Každý senzor má nějakou interní chybu měření, každý je rušený nechtěnými signály apod... Další stěžejní problém je pro nás proměnlivost prostředí a prostředí vůbec. Je velmi obtížné vytvořit algoritmy či systémy, které budou fungovat univerzálně v několika prostředích (což je potřeba například u autonomních vozidel), nebo dokonce i pro jedno konkrétní prostředí. A v neposlední řadě nám naši dokonalost narušuje nepřesný pohyb robota, který opět závisí na několika fyzikálních veličinách - tedy minimálně na typu podvozku (materiál, kinematika tělesa) a opět okolním prostředí (povrch po kterém se robot pohybuje, intenzita osvětlení).

Cílem této práce je vymyslet, navrhnout, implementovat a otestovat takovou fúzi dat z LIDARu[2] a stereo kamery, která by vylepšila metodu lokalizace či plánování trasy. Důležité je, že se chceme zaměřit na mapování (tvorbu mapy) z těchto senzorů. A dbáme především na to **jaké informace** a proč se budou spojovat, než jak. Na závěr pak ověříme, zda může být naše metoda přínosem či nikoliv. To vše na školním robotovi jménem RUDA.

V první části práce (2) se tedy seznámíme s různými přístupy lokalizace a mapování, které jsou nejpoužívanější, nebo je sama v této práci používám. Popisuji zde, jak takové metody pracují, co je pro ně klíčové a v čem se mohou lišit. Dále zde specifikuji hardware, se

[1] Je důležité si uvědomit hlavní rozdíl mezi asistenčním systémem - pouze napomáhá řidiči, popřípadě na okamžik převezme řízení, a autonomním systémem - funguje naprosto bez zásahu řidiče a musí být schopen se zotavovat ze svých chyb.

[2] Laser, přesněji Light Detection And Ranging.

kterým pracuji. Ve druhé části (3) pak popisuji současný stav problému, rozepisuji výhody a nevýhody dnešních přístupů, navrhuji zde a popisuji svůj vlastní přístup. Implementaci potom najdeme v kapitole 4, kde podrobně rozebírám celý systém modulů, které byly potřeba pro naši metodu vytvořit. Ve předposlední kapitole, kapitole Testování a vyhodnocení (5), se zabývám jednotlivými testy, rozebírám zde výsledky a občas diskutuji možná vylepšení. V závěru (6) pak shrnuji celou svoji práci a představuji zde možné úpravy a vylepšení mého přístupu do budoucna.

# Kapitola 2

# Používané metody a popis robota

## 2.1 Popis problému SLAM

Při zjišťování pozice robota se nemůžeme spoléhat pouze na jeho odometrii, viz sekce 2.1.1, protože jsou tato data relativně dost nepřesná a mohou obsahovat velkou chybu, která se s uraženou vzdáleností kumuluje. Proto používáme inteligentní senzory, které nám jsou schopny tuto pozici upřesnit. A toho využívá právě SLAM[1], což je základní prerekvizita pro autonomní mobilní robotiku. Jedná se o problém a jeho řešení, přičemž cíle jsou teoreticky dva, ovšem prakticky velmi úzce svázány.

Prvním cílem je vytvářet si **mapu okolí** v jednotlivých časových okamžicích (v textu budeme značit $m_t$), k čemuž robot používá různé senzory. Tyto senzory poskytují v jednotlivých časových okamžicích data (v textu budeme značit $z_t$) z aktuální pozice robota (v textu budeme značit $x_t$). Tato mapa je považována za hlavní výstup SLAMu. Za předpokladu, že získáváme všechny tři prvky s jakousi pravděpodobností, hledáme tedy

$$P(m_t|x_t, z_{1:t}), \tag{2.1}$$

kde $z_{1:t} = \{z_1, ..., z_t\}$.

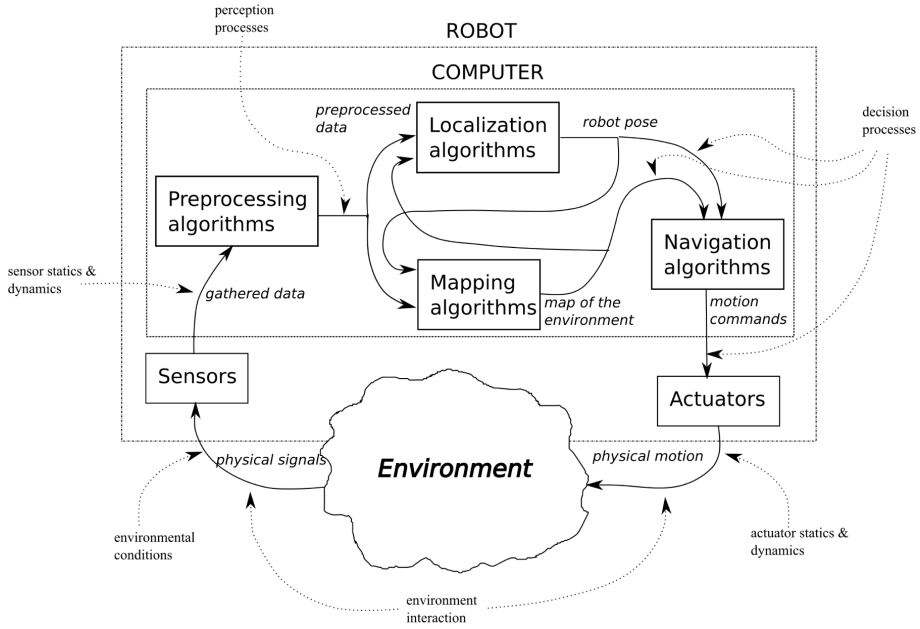
Naopak ale také potřebujeme robota v této mapě pomocí dat ze senzorů **lokalizovat**. Za stejného předpokladu jako u (2.1) hledáme tedy

$$P(x_t|m_t, z_{1:t}). \tag{2.2}$$

Z výše uvedeného můžeme vidět, že výpočet aktuální mapy $m_t$ a výpočet aktuální pozice robota $x_t$ jsou vzájemně závislé. Proto se tomuto přístupu říká **souběžná lokalizace a mapování** - potřebujeme znát mapu prostředí i pozici v této mapě v jednom okamžiku, přitom mezi sebou mají obě části zpětnou vazbu [9, 38].

Ilustraci vidíme na následujícím obrázku 2.1. Senzory (vlevo dole) získávají informace z okolního prostředí. Před vstupem do samotného jádra výpočtu je potřeba data nějakým způsobem předzpracovat. Provádí se například oddělení statických objektů od dynamických, ořezávání, filtrace samotných dat od šumu apod... Poté se dostáváme do výše zmíněného problému (prostřední část obrázku), kde můžeme vidět právě onu **zpětnou vazbu** z lokalizace na mapování a obráceně. Po získaném výstupu (mapa okolí a pozice robota) většinou navazuje na tyto algoritmy plánování trasy a navigace. Tím už se ale v této práci nezabývám. Po určení kam chceme s robotem dojet se dostávají ke slovu aktuátory, které provádí

[1] Simultaneous Localization And Mapping.




Obrázek 2.1: Řízení SLAM procesu. Nejistota (pravděpodobnost) se vyskytuje v každém kroce [9].

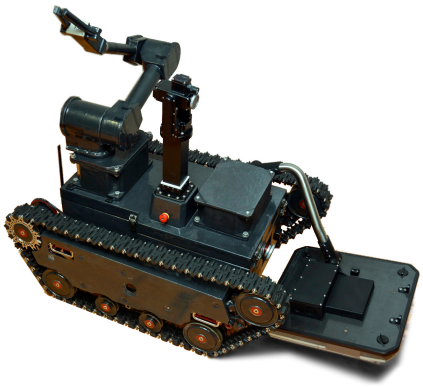
daný příkaz (otočení kola, přidání plynu), tedy intereagují s fyzickým prostředím a tím dostávají robota do dalšího stavu. Důležitým postřehem je, že v každé fázi tohoto procesu se setkáváme s nejistotou, která nastává u daných jevů a získaných informací. Cílem tedy nikdy není dosáhnout řešení, které bude za každé situace funkční, ale chceme řešení, které se bude co nejvíce blížit svojí uspěšností hodnotě 99,99%, viz [9] (především kapitoly 3, 4 a 5).

Duležitým aspektem při SLAM procesu je rychlost. Vyskytují se algoritmy, které jsou přesnější než ostatní na úkor časové složitosti. V těchto pravděpodobnostních přístupech můžeme vždy rychlost ovlivnit velice jednoduše. Takové algoritmy se potom řadí do tzv. offline zpracování. To ale většinou nechceme, chceme aby robot reagoval a pohyboval se v reálném čase. Zabývám se zde tedy pouze online přístupy.

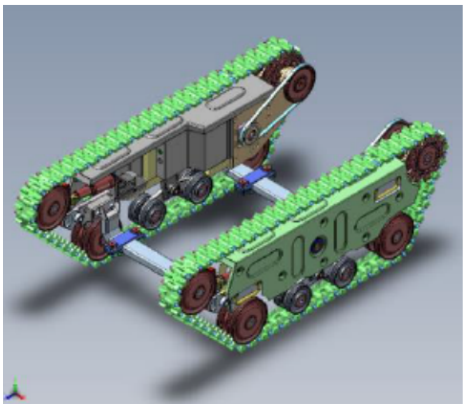
### 2.1.1 Odometrie a pohybový model robota

Jak už jsem zmínila výše, je to jeden ze vstupů do procesu SLAM. **Odometrie** je proces, který převádí data z enkodérů robota na jeho pozici a orientaci. Tuto pozici a orientaci ale dostáváme s nějakou nepřesností. Pokud by robot jezdil pouze v uzavřeném a nějakém řekněmě ideálním prostředí, mohl by si robot do určité míry pouze s odometrií vystačit. Realita je ale jiná, dostáváme roboty do fyzikálně proměnlivého a náhodného prostředí, kde už mezi pozicí měřenou odometrií a reálnou pozicí robota nastávají velké odchylky [40]. Základem odometrie je znalost geometrického (pohybového) modelu robota. Jedná se konkrétně o geometrii podvozku, abychom věděli, jakých pohybů je robot schopen. Popišme si tedy **pohybový model** našeho robota jménem RUDA. Robota vidíme na obrázku 2.2.

Tento typ robota se řadí mezi model tzv. tank či diferenciálně řízený robot. Základními prvky takového robota jsou dvě nezávisle poháněná kola/nápravy. Točí-li se stejným směrem stejnou rychlostí, robot se pohybuje přímo vpřed či vzad. Točí-li se kola stejnou rychlostí v opačných směrech, pak se robot otáčí na místě kolem svého referenčního bodu, což je



Obrázek 2.2: Pásový robot RUDA [26].



Obrázek 2.3: Detail podvozku našeho robota [26].

střed mezi nápravami. Pokud je ale rozdíl rychlostí náprav nenulový, ale obě nápravy se točí na stejnou stranu, pohybuje se robot po kružnici. Tento pohybový model je znázorněn na následujícím obrázku 2.4. Představme si tedy, že chceme získat novou globální pozici robota z $P_i = (x, y, \theta)$ do $P_{i+1} = (x', y', \theta')$. K výpočtu použijeme:

- poloměr kružnice $\mathbf{r_{center}}$ se středem v bodě $P$, po které se robot (referenční bod) pohybuje,
- poloměr kružnice, kterou opisuje levá náprava $\mathbf{r_{left}}$,
- poloměr kružnice, kterou opisuje pravá náprava $\mathbf{r_{right}}$,
- ujetá vzdálenost levé nápravy $\mathbf{d_{left}}$, referenčního bodu $\mathbf{d_{center}}$ a pravé nápravy $\mathbf{d_{right}}$.

Vzdálenost středu kružnice (bodu $P$) od referenčního bodu robota je dána poměrem obou rychlostí:

$$r_{center} = \frac{1}{2} L \frac{(v_{right} + v_{left})}{v_{right} - v_{left}},$$

kde $L$ je vzdálenost kol od sebe (rozchod) a $v_{left}$ a $v_{right}$ jsou rychlosti náprav. Přesný výpočet nové pozice $x'$ a $y'$ pak vypadá následovně:

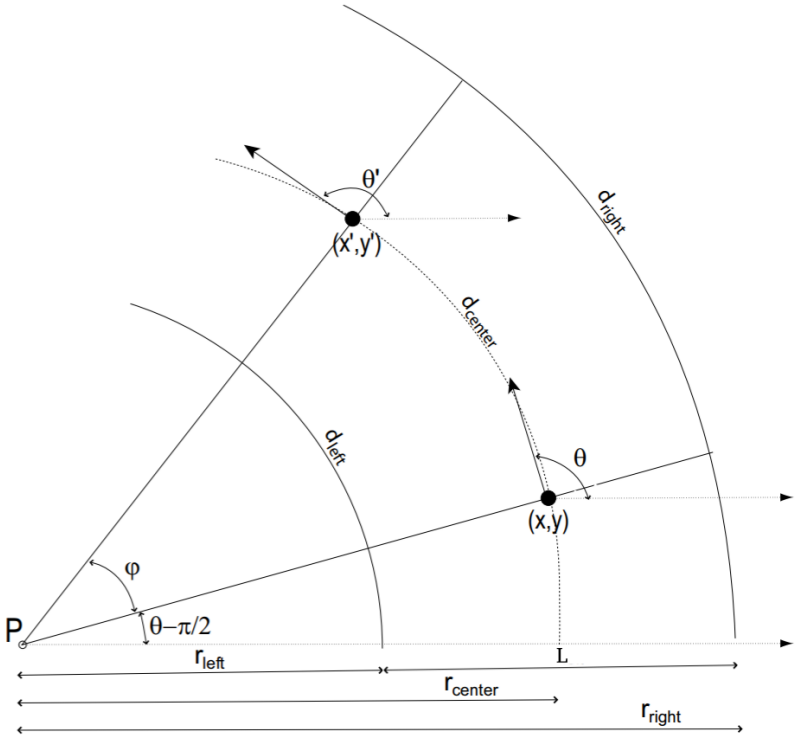
$$x' = x + r_{center}(-sin\theta + sin\phi cos\theta + sin\theta cos\phi) \tag{2.3}$$
$$y' = y + r_{center}(cos\theta - cos\phi cos\theta + sin\theta sin\phi) \tag{2.4}$$

Rovnice se dají zjednodušit díky předpokladu, že provádíme měření ve velmi malých časových usecích. Proto můžeme aproximovat $sin\phi = \phi$ a $cos\phi = 1$. Dostáváme tedy:

$$x' = x + r_{center}(-sin\theta + \phi cos\theta + sin\theta) \tag{2.5}$$
$$x' = x + r_{center}\phi cos\theta \tag{2.6}$$
$$x' = x + d_{center} cos\theta \tag{2.7}$$

a

$$y' = y + r_{center}(cos\theta - cos\theta + \phi sin\theta) \tag{2.8}$$
$$y' = y + r_{center}\phi sin\theta \tag{2.9}$$
$$y' = y + d_{center} sin\theta. \tag{2.10}$$




Obrázek 2.4: Pohybový model diferenciálního robota [24].

Pokud levé kolečko ujede vzdálenost $d_{left}$ a pravé $d_{right}$, změní se orientace robota o úhel $\phi$ následovně:

$$\phi = \frac{d_{right} - d_{left}}{L}, \tag{2.11}$$

kde $L$ je již zmíněný rozchod náprav. Celková ujetá vzdálenost se dá také vypočítat jako :

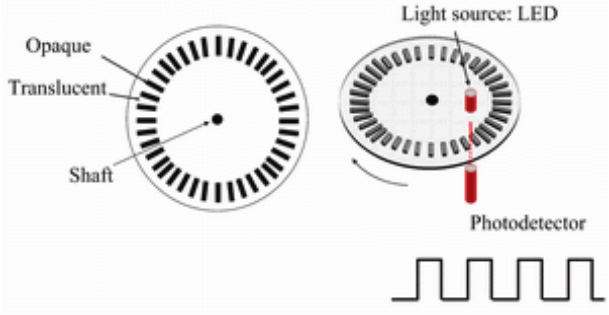
$$d_{center} = \frac{d_{left} + d_{right}}{2}. \tag{2.12}$$

Pak můžeme zapsat výsledný výpočet absolutní pozice $P_{i+1}$ jako:

$$P_{i+1} = \begin{bmatrix} x_i \\ y_i \\ \theta_i \end{bmatrix} + \begin{bmatrix} d_{center} cos\theta_i \\ d_{center} sin\theta_i \\ \phi \end{bmatrix}, \tag{2.13}$$

viz detailní výpočty v [24].

Dalším vstupem do procesu odometrie jsou **enkodéry**, které nám pomáhají kontrolovat nápravy (respektice jejich motory). Pří výpočtu odometrie totiž potřebujeme co nejpřesněji znát rychlost otáčení, abychom mohli co nejpřesněji určit právě polohu robota skrz jeho pohybový model, viz výše. V případě použití krokových motorů je možné předem přesně a řídit jejich pohyb, uhel natočení či rychlost otáčení bez nutnosti zpětné vazby. V našem případě jsou ale krokové motory nevhodné, proto u našich motorů potřebujeme zpětnou vazbu. Ta je realizována **rotačním optickým enkodérem**. Jeho princip je znázorněn na následujícím obrázku 2.5:




Obrázek 2.5: Princip rotačního optického enkodéru.

Enkodér obsahuje disk, který je v praxi připevněn na hřídel motoru. Na tomto disku jsou pak střídavě plochy, kudy (například) optický paprsek projde, nebo je pohlcen (odražen, atp. . . ). Optika může být realizována například tak, jako je vidět na obrázku. Z jedné strany disku je světelný zdroj, a na druhé straně disku přijímač tohoto světla. Světelný paprsek pak bud projde (a je detekován v přijímači), nebo neprojde. Světelné impulzy jsou pak převáděny na elektrické impulzy či obdelníkový signál (viz obrázek 2.5 vpravo dole). Pomocí tohoto signálu jsme pak schopni inkrementálně zjistit přesnou polohu hřídele [37].

### 2.1.2 Extrakce klíčových oblastí ze senzorových dat

Dalším vstupem do procesu SLAM jsou data z inteligentních senzorů. Z těchto dat potřebujeme extrahovat objekty či body, které splňují několik parametrů:

- jsou jednoduše znovu pozorovatelné,
- dostatečně odlišné od okolí,
- jsou dobře rozlišitelné mezi sebou,
- mělo by jich být co nejvíce
- a měly by být statické.

Například ve venkovním prostředí tedy chceme detekovat hrany či rohy budovy, lampy a sloupy. Naopak, co chceme vyřadit, jsou chodci a automobily. Na způsobu extrakce záleží podle typu senzoru, viz kapitola 2.2. Dále záleží na tom, co chceme v obraze detekovat.

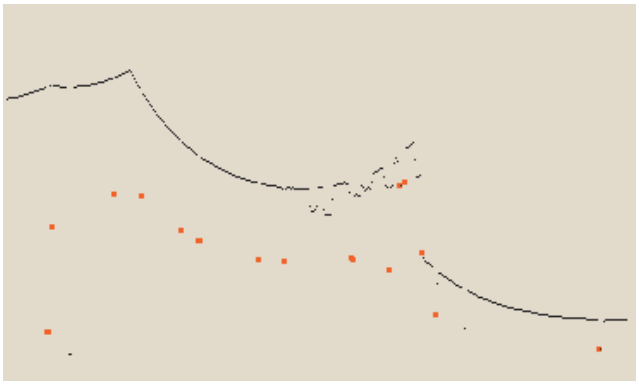
#### Klíčové oblasti z laserových dat

Co se týče laserových skenů, může detekce landmarků vypadat následovně. Pokud je cílem detekovat ve skenech extrémy, používá se metoda extrakce tzv. **Spikes landmarků**. Princip je zde takový, že se najdou hodnoty které se hodně liší od ostatních. Konkrétněji si můžeme představit například mřížová vrata. Pak se budou jednotlivé paprsky skenů chovat podobně jako na obrázku 2.6. Jak už jsem zmínila, tato metoda pouze hledá body, které se velmi liší od okolních bodů. Proto tento algoritmus selže v hladkém prostředí.

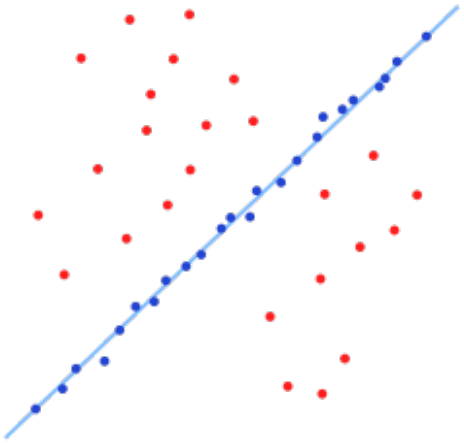
Jako další ukázku přístupu extrakce landmarků zde představím extrakci landmarků pomocí metody **RANSAC** (Random Sampling Consensus). Jedná se přesněji o extrakci přímek v laserových skenech. Oproti předchozímu případu zde chceme detekovat právě rovné / hladké plochy. Princip metody RANSAC pro nalezení přímky je založený na rychlém náhodném prohledávání prostoru za účelem získání vzorků, kterými prokládáme náš

požadovaný model (tedy v tomto případě přímku) a zjišťujeme, jak moc nám přímka odpovídá hledanému objektu. Bližší detail této metody můžete shlédnout v literatuře [11] a také v sekci 3.2.3, kde rozebírám podrobněji konkrétní aplikaci metody RANSAC pro detekci roviny v 3D prostoru.

Jako výsledek tohoto algoritmu se vybere přímka obsahující největší počet bodů splňujících podmínku definice přímky. Výsledek takového hledání vidíme na obrázku 2.7.



Obrázek 2.6: Extrakce Spikes landmarků. Červené tečky znázorňují extrahované landmarky [29].



Obrázek 2.7: Výsledek metody RANSAC.

Toto byly ukázky jen některých metod. V praxi se ale používají další, či spojení již zmíněných. Jedna z nejoblíbenějších metod je taková, kdy se za landmark jednoduše považuje každý bod v laserovém skenu [29, 38].

### Klíčové oblasti z dat kamery

Dále potřebujeme velmi často detekovat landmarky v obrazových datech kamery (o senzorech více viz kapitola 2.2). Zde už se dostáváme do rozsáhlé oblasti zpracování obrazu. Rozhodně už zde nemůžeme považovat za landmarky každý bod z dat kamery jako u LIDARu, neboť by zpracování takového množství pixelů bylo časově příliš náročné, přičemž by nám to žádné užitečné informace navíc nepřineslo. Používají se například metody SIFT (Scale-invariant feature transform) [21], SURF (Speed Up Robust Features) [2], nebo třeba jednoduchý **Cannyho hranový detektor** [25], jehož algoritmus si nyní přiblížíme a jeho výsledek vidíme na obrázku 2.9:

1. nejprve odstraníme šum z obrazu (předzpracování) - například Gaussovým filtrem s jádrem o velikosti 5, který může vypadat následovně

$$K = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 15 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}, \tag{2.14}$$

2. poté nalezneme gradient intenzity, a to například tak, že nejdřív aplikujeme na obraz dvě konvoluční masky - $G_x$ ve směru osy $x$ a $G_y$ ve směru osy $y$,

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}, \tag{2.15}$$

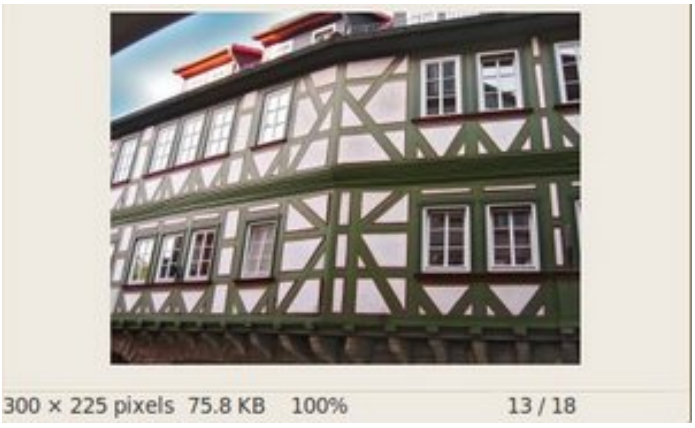
a pak můžeme určit velikost gradientu $G$ a jeho směr $\theta$ následovně:

$$G = \sqrt{G_x^2 + G_y^2} \qquad \theta = arctan(\frac{G_y}{G_x}), \tag{2.16}$$

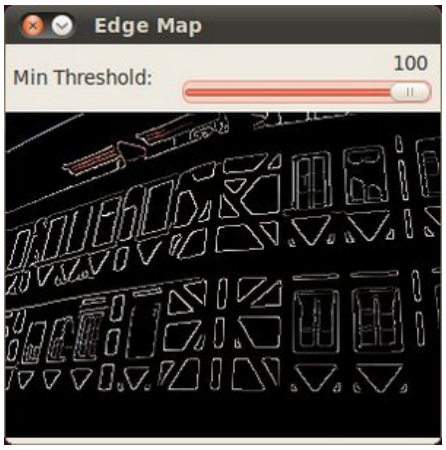
přičemž hodnotu úhlu můžeme zaokrouhlit na jednu ze čtyř hodnot - 0, 45, 90 či 135 stupňů,

3. hodnoty které nejsou maxima se potlačí, čímž se odstraní pixely, které nejsou považovány za hrany a vyniknou nám tenčí hrany

4. a na závěr se provádí prahování s hysterezí. Určí se prahy $T1$ a $T2$, mezi kterými může gradient kolísat. Pokud leží testovaná hodnota pixelu nad vyšším prahem, definitivně prohlásíme pixel za součást hrany. Pokud ovšem hodnota gradientu leží mezi prahy, pak označíme tento bod za hranový pouze tehdy, pokud už sousedí s nějakou hranou označenou dříve. Pixely s nižším gradientem jsou detektorem zamítnuty.




Obrázek 2.8: Zdrojový obrázek pro hranový Cannyho detektor.




Obrázek 2.9: Výsledná detekce hran Cannyho detektoru [25].

Cannyho detektor považujeme za výhodný hlavně z těchto tří důvodů: má nízkou míru chybovosti - v detekovaných hranách nalezneme pouze existující hrany, vzdálenost mezi detekovanými hranovými okrajovými pixely a pixely zdrojové hrany je minimální a navíc má minimální časovou odezvu - stačí jeden průchod našeho postupu [25].

Pro reálný venkovní svět je ale důležitou součástí přípravy klíčových oblastí také odfiltrování dynamických částí. Jelikož se z těchto dat později vytváří mapa, která slouží i pro opětovnou lokalizaci při průjezdu stejného místa. Pro data z kamery používáme například **trekování dynamických objektů pomocí optického toku** [28]. Pro LIDAR se používá například jednoduchá metoda metoda DATMO (detection and tracking of moving objects) viz [38].

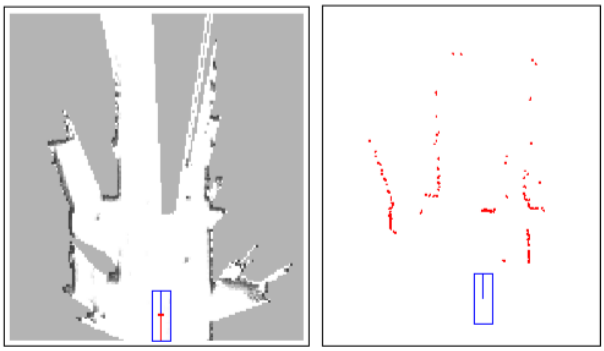
### 2.1.3 Asociace dat a tvorba mapy

Máme tedy detekované klíčové body / objekty, respektive landmarky (znázorňují objekty v reálném 3D světě), které jsme extrahovali z dat senzorů. Důležitým krokem a také velkým problémem je ale opětovné pozorování a **asociace těchto landmarků** v po sobě získaných senzorových datech. Prakticky dostáváme data ze senzorů v určitých časových okamžicích, přičemž se robot hýbe. Pokaždé jsme tedy schopni spočítat pozice klíčových oblastí, které jsou mezi jednotlivými časovými úseky vuči sobě posunuty o nějaký vektor. Tyto landmarky potřebujeme k sobě v každém kroce správně přiřadizovat. Zde se setkáváme s těmito problémy:

- ten samý landmark nemusí být pozorovatelný v každém kroce,
- můžeme detekovat landmark ale potom ho už nikdy nenajdeme,
- může nastat špatná asociace landmarků k landmarkům z předchozího kroku.

Nejen proto se zde opět počítá s pravděpodobnostmi. Například klíčový bod prohlásíme landmarkem až poté, co ho detekujeme několikrát. Naopak znovu nedetekované landmarky smažem. Konkrétní příklad s počítáním takových pravděpodobností můžete shlédnout v kapitole o laser SLAMu (2.2.2).

Z landmarků se vytváří **mapy**. Ty mohou být reprezentovány různě. Znovu si uvědomme, že za landmark může být považován každý senzorem detekovaný objekt. Tak tomu také velmi často (konkrétně u laserových dat) bývá a takovým mapám se říká mapy **metrické**. Tyto mapy obsahují přímá měření ze senzorů (pravděpodobnostní hodnoty), která jsou rozložena do pravidelné mřížky. Taková mapa často vypadá následovně:



Obrázek 2.10: Ukázka metrické mapy (vlevo) vytvořené ze skenů laseru (vpravo) [38].

Detaily k metrické mapě budou popsány v kapitoly 2.2.2. Toto byla ukázka 2D metrické mapy. 3D skenováním ale můžeme vytvářet i 3D mapu prostoru. Pokud bychom ale chceme počet buněk v mapě zhruba kvadraticky zvětšit, musíme už zavést do výpočtů optimalizace. Pro tyto případy se vytváří stromové struktury, viz [41].

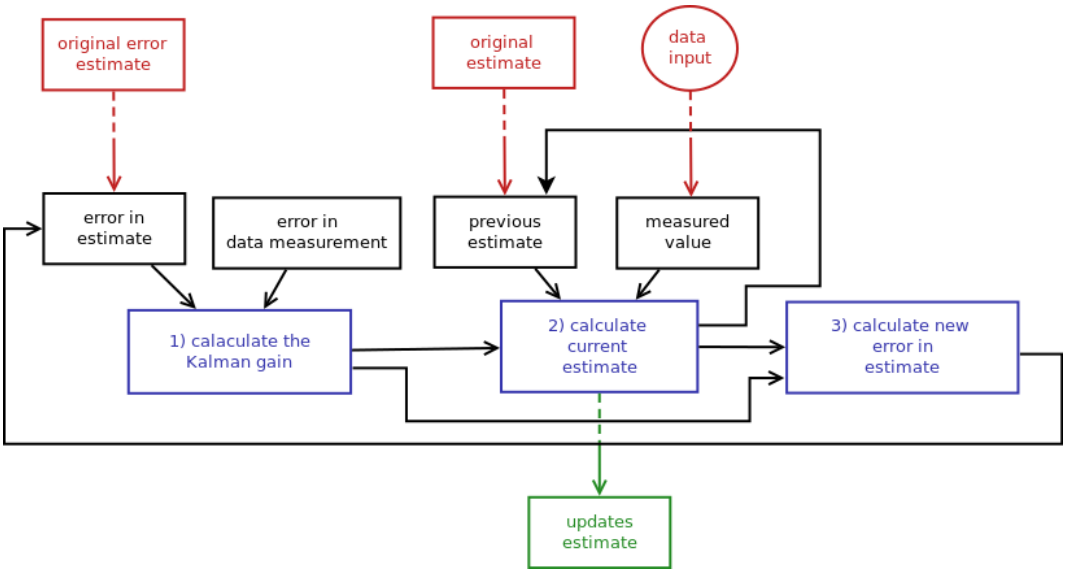
Další oblíbeným typem map bývají mapy **topologické**. Zde není důležitá přesná pozice landmarků, ale zajímá nás jejich vzájemné propojení. Můžeme se zde například inspirovat metodou Growing Neural Gas [30]. Z používaných typů zmíním ještě mapy **geometrické**, které jsou složeny z geometrických primitiv, dále 3D modely, mapy hybridní, atp...

### 2.1.4 Lokalizace robota

Pokud sledujeme informaci o pozici robota pouze odometrií (tedy o relativní změně v čase), chyba se akumuluje a je tedy potřeba tuto informacit upravit / zpřesnit. A právě zde se dostáváme k problému současné lokalizace a mapování. Robot se nám posune do neznámé pozice $x_t$ (změna v odometrii). V této pozici dostaneme na vstup data ze senzorů $z_t$ a chceme pomocí těchto dat a aktuální mapy $m_{t-1}$ určit pozici robota v mapě a mapu aktualizovat pomocí aktuálních dat ze senzorů a aktuální pozice. Tady vidíme přesně to, co jsem popsala v kapitole 2.1. Všimněme si, že aktuální mapa k dispozici je z času $t-1$, protože nová mapa v čase $t$ ($m_t$) vzniká až po lokalizaci robota. Pro co nejpřesnější odhad pozice robota v mapě se používají následující uvedené přístupy.

#### Kalmanův filtr

Kalmanův filtr (KF) - jedná se o matematický iterativní proces, který používá množinu rovnic (často v maticovém zápisu) pro velmi rychlý a co nejpřesnější odhad (v našem případě) pozice a to na základě aktuálně měřených dat, které obsahují nejistotu, náhodnou chybu, proměnlivost... Jelikož je kalmanův filtr relativně oblíbený, ukažme si alespoň jednoduché schéma, jak takový filtr funguje.




Obrázek 2.11: Ukázka principu Kalmanova filtru pro 1 vstup.

Za celým procesem filtru se skrývají 3 rovnice, kterými proces iteruje za cílem odhadnout co nejpřesněji správnou hodnotu (v našem případě pozici). A to výpočet tzv. Kalmanova přírůstku, aktualizace nového stavu (pozice) a přepočet chyby v odhadu - neboli jakousi nejistotu v určení aktuální pozice (všechny 3 rovnice zastupují právě modré oblasti v obrázku 2.11). Vidíme, že na vstupu výpočtu přírůstku máme jak chybu odhadu, tak chybu měření. Tato část filtru je schopna určit, jak moc je měření přesné vůči odhadu a poté nastavit hodnotu přírůstku tak, aby správně ovlivnila další průběh rovnic. Kalmanův přírůstek (**Kalman Gain** - $KG$) spočítáme následovně:

$$KG = \frac{E_{est}}{E_{est} + E_{mea}}, \tag{2.17}$$

kde $E_{est}$ je chyba odhadu a $E_{mea}$ je chyba měření, a výsledek této rovnice bude v intervalu $< 0, 1 >$. Na to potom navazuje právě **výpočet (odhad) aktuální pozice** $EST_t$ robota, který má na vstupu Kalmanův přírůstek, předchozí stav a aktuální měřenou hodnotu. Zde se právě využívá Kalmanova přírůstku tak, že čím je měření přesnější, tím větší klademe důraz na rozdíl mezi měřením a odhadem. Obráceně pokud byl přesnější odhad, bude ve výpočtu dominovat složka odhadu (a rozdíl meření a odhadu už má menší váhu). Výpočet vypadá následovně:

$$EST_t = EST_{t-1} + KG[MEA - EST_{t-1}], \tag{2.18}$$

kde $EST$ je hodnota odhadu a $MEA$ je naměřená hodnota senzory. Na závěr každého cyklu spočítáme novou **chybu v odhadu** $E_{EST_t}$:

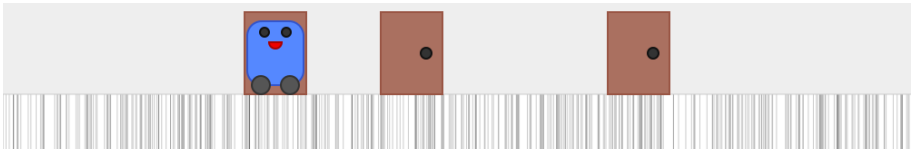
$$E_{EST_t} = [1 - KG](E_{EST_{t-1}}). \tag{2.19}$$

Takovýto Kalmanův filtr ale funguje pouze pro lineární systémy. V praxi se tedy v robotice používá spíše rozšířený Kalmanlv filtr, protože jsou většinou tyto systémy nelineární. Kalmanův filtr je velmi rozsáhlé téma. Podrobnosti můžete shlédnout například v [12, 3]

### Částicový filtr

V dnešní době je ale nejvíce používanou metodou lokalizace pomocí částicového filtru. Charakteristickou vlastností této metody je způsob reprezentace hustoty pravděpodobnosti popisující odhad pozice robota. Využívá tzv. vzorků, jejichž váhy a rozložení ve stavovém prostoru určuje, jak moc je která pozice pravděpodobná. Algoritmus se skládá ze tří klíčových kroků opakujících se ve smyčce:

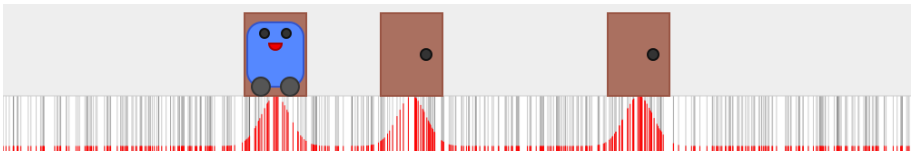
1. **Predikce** - ještě před začátkem celého algoritmu se určí počet částic $N$, které se budou vždy v tomto kroce predikce znovu generovat / aktualizovat. Tato sada vzorků je vygenerována právě na potenciálních pozicích robota (kde by se mohl robot vyskytovat). Vždy se každá částice posune například s odometrií robota.



Obrázek 2.12: Predikce pozice robota.

   V dolní části obrázku (svislé čáry na bílém pozadí) vidíme rozložení částic (potenciálních pozic robota). Obrázek znázorňuje první krok, nemáme tedy žádné informace o pozici robota. Proto jsou vzorky vygenerovány rovnoměrně po celé ploše.

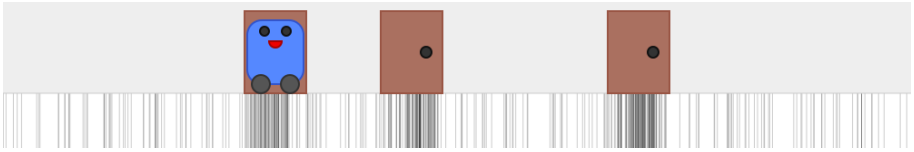
2. Následuje úprava váhy každé částice. Tomuto kroku se říká **korekce**. Z každé vygenerované částice (pozice) v předchozím kroce nyní vyzkoušíme, jak moc aktuální pozorování (konkrétně landmarky) sedí na landmarky v naší doposud vytvořené mapě. Některé částice tak dostanou větší důvěryhodnost, že jsou právě ony na naší hledané pozici. Ukázku tohoto kroku vidíme na obrázku 2.13. Výška červených čar v oblasti rozprostřených částic říká, jak moc je daná pozice pravděpodobná (zde navíc vidíme,



Obrázek 2.13: Korekce pozice robota.

že máme nalezena 3 maxima - tedy 3 polohy, kde by se robot podle doposud získaných dat mohl nacházet).

3. Posledním krokem je **převzorkování**. Kolem částic / shluků s největší váhou se opět vygeneruje všech N částic (respektive se vzorky s malou váhou smažou a vzorky s velkou váhou se rozdělí). Výsledek převzorkovaných částic vidíme na následujícím obrázku 2.14.



Obrázek 2.14: Převzorkování částic filtru.

Výhod má tento přístup hned několik. Především to, že reprezentace polohy pomocí vzorků neklade důraz na žádný konkrétní tvar hustoty pravděpodobnosti a navíc jsme schopni detekovat více než jedno maximum (robot se tedy může vyskytovat velmi pravděpodobně na vícero místech) [35, 39].

Pro jednoduché prototypování se občas používá velmi ořezaná metoda částicového filtru. V každém kroce testujeme pouze ty pozice, které jsou dostupné z naměřených hodnot senzorů přepočítaných na odometrii plus mínus nějaká odchylka. Provádí se zde také jakási predikce, nicméně namísto korekce už se vybere přímo pozice, která se prohlásí za aktuální pozici robota. Nenásleduje ani žádné převzorkování. Takového přístupu se využívá zpočátku vývoje nějakého algoritmu, přičemž máte vhodná testovací data. Tento jednoduchý algoritmus uvažují například v [38].

Podle toho, co je pak v konkrétním SLAMu klíčové (často jsou to právě filtry), pak dostávají algoritmy své názvy: EKF SLAM (použití rozšířeného kalmanova filtru), FAST SLAM (použití částicového filtru), Graph SLAM (reprezentuje pozice grafem), atd. . .

### 2.1.5 Zdroje chyb

V posledních 20 letech se v robotice používá pravděpodobnost už snad v každém výpočtu. A to především proto, že tato situace odpovídá reálnému světu, kde nám právě pomáhá řešit výpočty, které jsou vždy plné nechtěných informací. Chyby se tedy mohou vyskytovat například v těchto místech:

1. **hardwarové komunikační prvky** - špatné napětí řídícího signálu, výpadky,
2. **realizace senzorů** - každý senzor má svoji interní chybu, nikdy není zkonstruován dokonale, okolní jevy mají často negativní dopad na data ze senzorů - příliš velké osvětlení pro kameru, lesklé povrchy pro LIDAR, zkrácení, zrcadlový odraz či přeslech u sonaru,

3. **diskretizace** - převod reálného světa do diskrétního světa bitů,
4. **neideální prostředí** - podkluzování kol na ledu, což nám zanáší chybu do odometrie, hustý déšť či sněžení pro LIDAR,
5. nechtěné otřesy senzorů při měření, zpoždění, u složitějších systémů pak kalibrace a mnoho dalších...

Pravděpodobnost nám poskytuje jakési rozhraní, které nám pomáhá se s těmito zdroji chyb vypořádat.

## 2.2 Senzory v problematice SLAM

Ve SLAM přístupu se využívají různé senzory, přičemž vždy záleží na situaci, pro kterou je chcem použít. Rozhodně se ale vždy jedná o senzory, které jsou schopny měřit jakýmkoli způsobem vzdálenost. Pod vodou se často používá sonar, na cestách to bývá radar a GPS. Nejčastěji jsou ale pro naše účely využívány LIDAR (Light Detection And Ranging) a kamera. Dnes už je dostupný i 3D LIDAR, jehož cena je ale velmi vysoká (statisíce). Dokáže ale poskytnout velmi důležitá a vzácná data. Tyto dva senzory (2D LIDAR) bývají doplněny o GPS senzor. Ten je v dnešní době už velmi přesný. Tento senzor nám může výsledky jen podpořit, nicméně spolehnout se na něj jako na klíčový senzor nedá. Ne vždy je totiž GPS signál dostupný. Nyní se budeme věnovat senzorům, které i já ve své práci používám.

### 2.2.1 Použité senzory a jejich parametry

Robot RUDA je osazený mnoha senzory. Pro svůj účel ale používám právě tyto dva: LIDAR a stereo kameru.

#### LIDAR

Lidar se používá především proto, že dokáže poskytnout velmi přesná data a to na vysoké frekvenci. Dále jsou tato data velmi jednoduše zpracovatelná. Nevýhodou bývá cena, která je u těchto zařízení relativně vysoká (desítky tisíc).

Verze LIDARu robotu RUDA je **SICK lms111-10100**. Jeho podstatné parametry jsou následující [34]:

- je vhodný i pro venkovní použití,
- úhel záběru je 270°,
- skenovací frekvence buď 25 Hz nebo 50 Hz,
- úhlové rozlišení $0,25°$ nebo $0,5°$,
- provozní vzdálenost od překážek $0,5 - 20$ metrů,
- maximální vzdálenost vhodná pro měření 18 metrů,
- zpoždění $\leq 20$ milisekund,
- a důležitá rozhraní pro komunikaci - sériové (RS-232), ethernet a CAN (Controller Area Network).


Obrázek 2.15: Můj pracovní senzor LIDAR od výrobce SICK [34].

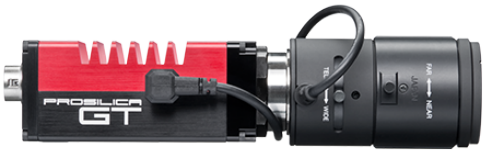
**Princip mechaniky uvnitř LIDARu** je následující. Celé měření probíhá v jedné rovině (rovina je rovnoběžná s černým horním víkem na obrázku 2.15). Do této roviny se promítá jeden laserový paprsek (například infračervené záření - 905nm). Ten je namířen na vnitřní zrcátko, které se otáčí (rozdíl úhlu mezi jednotlivými natočeními je právě úhlově rozlišení). Každý takto vyslaný paprsek je odražen (v případě úspěchu) a zachycen zpět na detektoru světla. Otáčením zrcátka vnášime mezi jednotlivé vysílání paprsků miniaturní zpoždění, o kterém je dobré býti informován.

### Stereo kamera

Další senzor, který používám, je stereo kamera. Ta se používá proto, že je dnes počítačové vidění velmi vyspělé a dá se z obrazu kamery leccos zjistit. Dále samozřejmě proto, že se jedná o relativně lený senzor po finanční stránce oproti ostatním.

Konkrétní model, se kterým pracuji, je stereokamera složená ze dvou kamer **Prosilica GT 1290**. Důležité parametry jsou [33]:

- rozlišení 1280x960,
- typ senzoru CCD Progressive,
- snímací frekvence 33,3 fps,
- 14 bitový AD převodník,
- 128 MByte RAM,
- RGB, YUV výstupy, atp...




Obrázek 2.16: Kamera, která je použita ve stereo kameře [33].

### 2.2.2 Laser SLAM

Obecný princip metody SLAM jsme si popsali v textu výše, a teď se můžeme podívat na klíčové odlišnosti použití různých senzorů. Ty spočívají především v použití různých landmarkových map. Mezi nejoblíbenější mapu z LIDARových dat patří tzv. **Occupancy Grid**. Jedná se o reprezentaci mřížky, která se skládá z diskrétních buněk o nějaké velikosti. Náhled na takovou mapu vidíte na obrázku 2.17.

Obrázek 2.17: Pravděpodobnostní mřížková mapa z LIDARu.

Mapa je tvořena pravidelnou mřížkou, do jejíž buněk je mapovaný reálný prostor (1 buňka odpovídá například rozměrům 20x20 centimetrů). Každá taková buňka uchovává pravděpodobnost (tedy hodnotu $0 - 1$), zda je překážkou nebo ne. Rozlišujeme 3 základní oblasti:

- oblast okupována překážkami - černé nebo velmi tmavé buňky (například hodnoty pravděpodobností $P > 0.5$),
- oblast bez překážek - bílé nebo velmi světlé buňky (hodnoty pravděpodobností $P < 0.5$)
- a prostředí o kterém nejsme schopni nic rozhodnout (šedé vnější oblasti, oblasti za překážkami, $P = 0.5$).

Do těchto buněk zapisujeme hodnoty podle aktuálního skenu, jehož příklad vidíme na obrázku 2.17 vpravo. Když dostaneme tato data z LIDARu, přičteme vhodné pravděpodobnosti do buněk v mapě a tím je **mapa aktualizována**.

Pro aktualizaci mapy ale nejprve potřebujeme získat co nejpřesnější pozici robota. Tu můžeme vypočítat například tzv. **matchováním skenů** - zasadíme sken do mapy z nějaké pozice a provádíme testy jak moc tento sken sedí na mapu z předchozího kroku. Tímto přístupem vyhrává sken, kde se překrývá nejvíce černých bodů v mapě a získáváme odhad pozice [38].

### 2.2.3 Kamera SLAM

S kamerou už je tvorba mapy o něco složitější. Mapa je tvořena ve 3D prostoru a navíc z dat kamery nemáme tak jednoduše dosažitelnou informaci o hloubce (vzdálenosti) objektů jako u LIDARu, který nám právě hloubkovou informaci dává na prvním místě. Když tedy extrahujeme klíčové body, chceme z nich udělat landmarky. To znamená, že chceme jejich pozici určit ve 3D světě a nejen ve snímku kamery. K tomu nám pomůže například zpětná 3D projekce a optický tok [43]. Pro takový přístup ale potřebujeme více snímku, abychom byli schopni hloubku objektů odhadnout. Nastávají zde i problémy s inicializací. Proto se často používá **stereo kamera**, kde jsme schopni se znalostí geometrie obou kamer získat hloubkovou informaci hned v prvním kroce, viz získání informací o disparitě a tvorbě hloubkové mapy například v [7, 15].

Propagační charakter takové mapy může vypadat například tak, že se nebude vybírat pouze nejlepší pozice robota a landmarků, ale zase se zde bude počítat s nějakým pravděpodobnostním rozložení v blízkém okolí nejlepších pozic, kde by se tyto objekty mohli nacházet. Ukázku takové mapy vidíme na obrázku 2.18 a):

(a) (b)

Obrázek 2.18: a) Pravděpodobnostní 3D mapa landmarků b) Klíčové objekty detekované v obraze a jejich zpětná projekce do souřadnic světa [6].

Na obrázku 3D mapy vidíme barevné **elipsoidy** a kameru. Tyto elipsoidy značí, kde se vyskytuje landmark (tedy někde uvnitř). Červené jsou naměřené landmarky v aktuálním kroce, žluté jsou pozice landmarků, které ale v aktuálním kroce nejsou změřeny. Dále se do mapy ukládají i landmarky, které vznikly díky chybnému měření - jsou již odstraněny z mapy. Detaily tohoto přístupu jsou například v [6].

# Kapitola 3

# Současný stav problému a popis vlastního přístupu

V praxi se ovšem nepoužívá pouze jeden konkrétní senzor, ale několik. Je potřeba myslet na to, že ne vždy může být každý senzor dostupný, že v některých prostředích jsou nám určité typy senzorů k ničemu, a že více senzorů by mělo poskytovat větší přesnost. Proto si ukážeme některé přistupy, které se dnes používají a zhodnotíme jejich přínosy a naopak negativní vlastnosti.

## 3.1 Kritika současného stavu

**Fúze hotových SLAMů** - jedna z možností fúze těchto dvou senzorů je taková, že necháme nezávisle na sobě pracovat SLAM jak z kamery, tak z LIDARu. V tom případě máme lokalizaci i mapování individuálně u každého senzoru a stačí nám například přepínat mezi prioritami v závisloti na změně prostředí. Pokud se tedy bude robot pohybovat v prostředí, kde se nachází hodně objektů v dosahu laseru, stačí nám nastavit vysoká priorita právě pro LIDAR SLAM. Ten je také mnohem přesnější než kamera. Pokud se naopak bude robot pohybovat v prostředí, kde jsou objekty daleko od robota či nad / pod oblastí viditelnosti laseru, je vhodné přepnout větší prioritu na kamera SLAM, který se v této situaci bude orientovat lépe.

Výhodou je, že spousta takovýchto SLAMů pro jednotlivé senzory už je napsaná a otestovaná, viz například pro kameru [6] a pro laser [38]. Nevýhodou ale je, že nám tím přibývá další rozhodující článek navíc, což může být pro výše zmíněné detektor prostředí. Navíc jsou data z kamery mnohem více nepřesná než z LIDARu a v 99% případů je zbytečné, aby stroj počítal paralelně LIDAR slam i kamera SLAM, když se skoro vždy robot lokalizuje na základě LIDARu. Další nevýhodou je, že kromě nastavování již zmíněných priorit se s hotovými SLAMy moc dále upravovat nedá. Otázkou dále zůstává, zda je lepší využít dvě kamery jako stereo vizi pro hloubková data, nebo pro dva mono SLAMy.

**Fúze na urovni dat** - spojování dat o úroven níže je v praxi užitečnější. Tím nejčastěji myslíme tvorbu mapy do jednoho SLAM přístupu z více senzorů. Můžeme za základ považovat například 3D mapu z 3D LIDARu a doplnit ji informacemi z kamery. Taková fúze je ale vhodnější spíše pro roboty, kteří jsou schopni pohybu ve třech osách. Dále je dnes 3D LIDAR stále vzácný, takže nemusí být vždy dostupný. Ale o to je to modernější a aktuálnější téma.

Pro nás je ale zajímavější do základní 2D laserové mapy přidání informace ze stereo kamery. Výzkumů v tomto oboru už bylo také mnoho. Například v [22, 19] se zabývají detekcí objektů ze stereo vize. Přesněji je princip založen na **detekci pozemní roviny v disparitní mapě**, ve které se následně určují překážky. Autoři používají tzv. **v-disparitu** - histogramový obraz. Každý řádek v-disparitního obrazu je histogram různých disparit, které se vyskytují na tomtéž řádku v disparitní mapě. V ideálním případě se pak body ležící v rovině v disparitní mapě projeví ve v-disparitě jako výrazná přímka [19]. Taková metoda ale selže, pokud překážky okupují většinu části snímku (roviny). Pak se přímka definující pozemní rovinu ve v-disparitě deformuje a není již přímkou. Další výzkum [22] pojednává například o detekci překážek ze stereo vize tím způsobem, že nejprve transformuje disparitní mapu na body ve 3D prostoru v ortogonální projekci systému kamery a až poté použije hloubkovou informaci pro odhad pozemní roviny. Na závěr proběhne detekce překážek. Naopak v [19, 10] využívají hloubkovou informaci pro odhad roviny přímo z disparitní mapy. Kvůli přenosu co nejmenší chyby je vhodnější získávat informace přímo z disparitní mapy, protože se při transformaci bodů z disparitní mapy do 3D ortogonálního prostoru používají nelineární transformace, kde se používají odhadnuté parametry kamery. Tyto parametry mohou vzniklé chyby při odhadu (přesněji kalibraci) dále propagovat.

## 3.2 Popis vlastního přístupu

V této práci otestujeme vlastní přístup, kde se do 2D pravděpodobnostní mřížkové mapy z LIDARu snažíme přidat informaci ze stereo kamery (kterou také přemapujeme do 2D). Taková mapa nám pak může pomoci jak při lokalizaci (tedy konkrétně SLAM přístupu), tak při plánování trasy. Metodu jsme vybrali z **několika důvodů**:

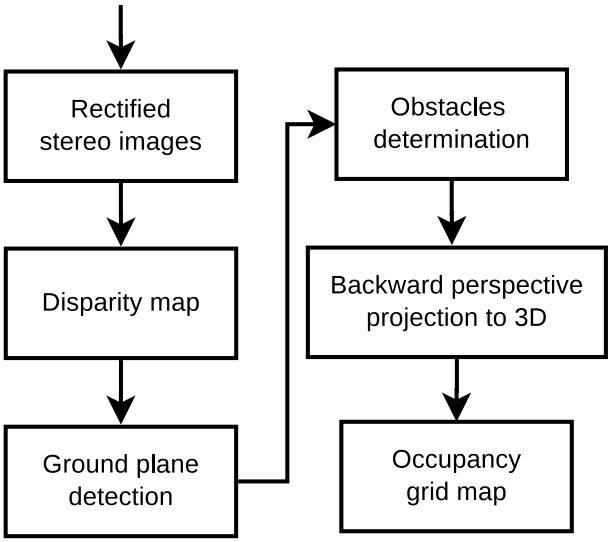
- vzhledem k tomu, že LIDAR poskytuje informaci pouze v jedné rovině, je vhodné ji doplnit informací z kamery ohledně bližšího okolí,
  - přínosu pro lokalizace dosáhneme tehdy, pokud se LIDAR nebude moci chytit dostatečného množství objektů pro mapování,
  - přínosu pro plánování trasy dosáhneme tehdy, když LIDAR uvidí jen část překážky, která nepředstavuje ohrožení pro robota, ale kamera vidí, že objekt je prostorově rozsáhlejší a překážkou být může,
- mapa z LIDARu je vhodný základ, protože je velmi přesná,
- náš robot provádí pohyb pouze ve 2 osách, tedy výškovou informaci od jisté míry nepotřebujeme,
- tyto senzory jsou v mobilní robotice nejpoužívanější,
- náš robot RUDA je jimi také osazen.

Na obrázku 3.1 vidíme vyobrazeny situace, ve kterých by náš přístup mohl být přínosem. První snímek je ukázkou, kdy nastává problém při mapování a lokalizaci pouze LIDARem. Pokud totiž laser nemá dostatečnou vzdálenost snímání, uvidí jen dvě rovnoběžné překážky - tedy levou a pravou stěnu. Když se pak robot bude pomocí matchování skenů (viz kapitola 2.2.2) lokalizovat, bude umět určit úhel natočení, ale přesnou pozici vpředu / vzadu mezi stěnami nikoliv. Kamera může zachytit objekty na stěnách a tím polohu robota upřesnit. Další případ vylepšení vidíme na prostředním obrázku. Zde může být přidání kamery

Obrázek 3.1: Situace, ve kterých by naše metoda mohla být přínosem.

užitečné proto, že LIDAR vidí pouze v jedné rovině, tudíž nevidí za překážky. Oblast za modrou překážkou v levém dolním rohu by tak označil za neprozkoumanou. Kamera ale vidí, že za modrou překážkou je volno, takže podle ní může tudy plánovač klidně určit trasu. Poslední užitečná situace v našich příkladech je taková, kdy laser je schopný vidět pouze nohy od stolu, ale ne stůl jako celek. Proto si plánovač (pouze z dat laseru) může myslet,že mezi nohama robot projede, přičemž deska stolu představuje pro robota překážku.

Za prerekvizitu tedy považujeme mapování (SLAM) z LIDARu. Tato technika je dnes kvalitně otestována a osvědčená, viz sekce 2.2.2. Do 2D mapy z laserového senzoru se dále snažíme přidat informaci ze stereo kamery. Z důvodů zmíněných v sekci 3.1 jsme zvolili přístup, ve kterém zpracováváme informace přímo z disparity. Schéma můžeme vidět na obrázku 3.2:




Obrázek 3.2: Schéma vlastní metody získání informací ze stereo vize.

Z rektifikovaných stereo snímků získáme **disparitní mapu**. V této mapě pomocí metody **RANSAC** (Random Sample Consensus) a **metody nejmenších čtverců** (LSQP - Least Squares Plane) detekujeme pozemní rovinu, po které se robot pohybuje. Stále na úrovni disparitní mapy pak rozlišíme, které oblasti představují překážku pro robota a které jsou volné. Teprve potom provádíme **zpětnou perspektivní projekci** za získáním pozic objektů v 3D ortogonálním systému kamery (robota). Tyto objekty (body) na závěr transformujeme do 2D souřadnic mapy LIDARu.

### 3.2.1 Získání rektifikovaných snímků

Při snímání scény dvěma kamerami můžeme ze snímků získat hloubkovou informaci či zrekonstruovat původní 3D scénu. Přímá data z kamery ale mohou být deformována (soudková distorze, rybí oko) například kvůli nepřesnosti čoček. Dále potřebujeme ve stereo vizi znát přesnou vzájemnou polohu kamer včetně parametrů kamer samotných. K těmto účelům nám poslouží **kalibrace** - proces, při kterém získáváme vnitřní a vnější parametry kamer. Při stereo vizi pak i vzájemnou polohu kamer. **Vnitřní kalibrační parametry** (intrinsic) kamer jsou:

- $(x_0, y_0)$ - počátek souřadného systému (v pixelech) v obraze kamer, většinou bývá blízko středu obrazu,
- $f$ - ohnisková vzdálenost kamery
- $s$ - zkosení světločivých buněk na CCD čipu (většinou jsou bunky čtvercové),
- $a_x/a_y$ - poměr stran světločivých buněk (většinou je poměr roven jedné).

Vnitřní parametry mimo jiné bývají známé i přímo z výroby. **Vnější kalibrační parametry** (extrinsic) nám popisují pohyb kamery ve statické 3D scéně, nebo naopak popisují pohyb objektů kolem kamery. Jedná se o matici $[R|t]$, kde $R$ značí rotaci a $t$ translaci. Pak se souřadnice bodu $(X, Y, Z)$ přepočítají do systému kamery jako:

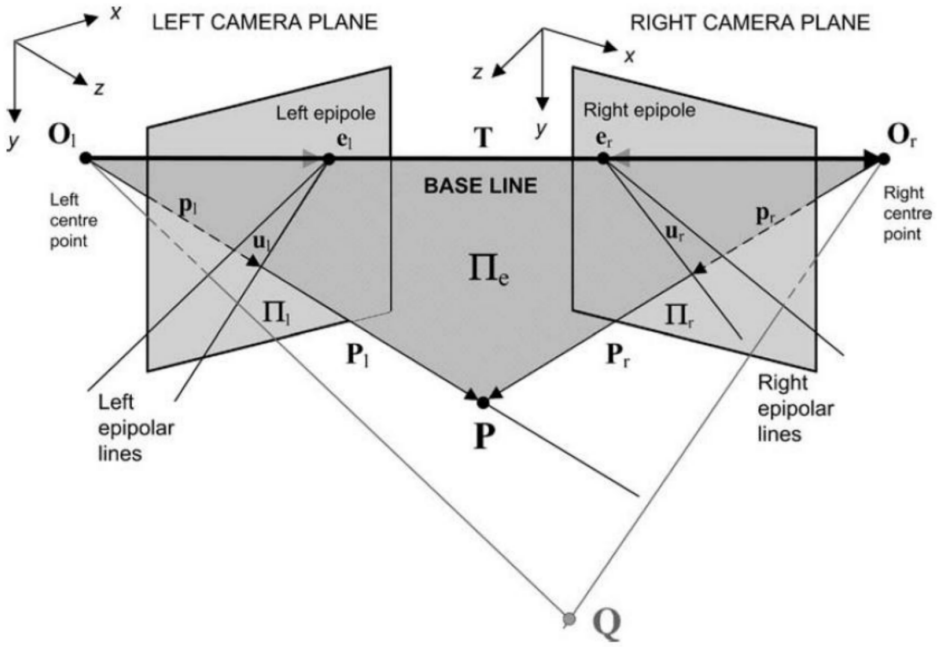
$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t. \tag{3.1}$$

Jelikož ve stereo vizi používáme více kamer (v našem případě dvě), musíme znát jejich vzájemnou polohu. Stereo kalibrace je odvozena od základu **epipolární geometrie** (podrobněji rozebráno například v [16]). Jedná se o vnitřní projektivní geometrii mezi dvěma pohledy kamer, přičemž kalibrace je nezávislá na struktuře scény, ale závisí na vnitřních a vnějších parametrech kamer. Jestliže máme dva pohledy kamer a matice $F$ je matice, která definuje geometrii scény a je o rozměru 3x3 a bod v prostoru $X$, který je zobrazen v pohledu jedné kamery jako $x$ a v pohledu druhé kamery jako $x'$, pak platí následující vztah:

$$x'^T F x = 0. \tag{3.2}$$

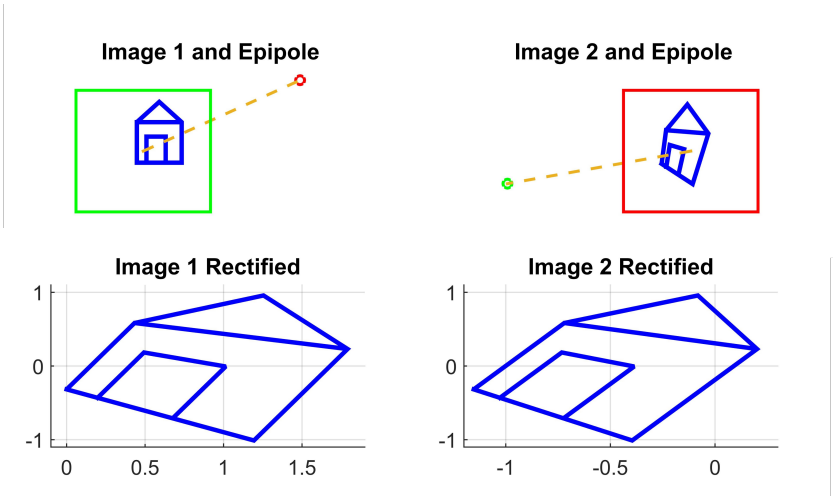
Na obrázku 3.3 vidíme vlastnosti epipolární geometrie, kde jsou vyznačeny následující důležité části:

- **optické středy kamer $\mathbf{O_l}$** a **$\mathbf{O_r}$**,
- **epipolární rovina** $\Pi_e$ je rovina, která je určena základní linií (baseline) a libovolným bodem v prostoru $(P, Q)$,
- **epipóly $\mathbf{e_1}$** a **$\mathbf{e_2}$** jsou místa na projekčních rovinách, kudy prochází spojnice mezi kamerami (baseline),
- **epipolární linie $\mathbf{u_l}$** a **$\mathbf{u_r}$** jako průnik epipolární roviny $\Pi_e$ s projekční rovinou kamery $\Pi_l$ nebo $\Pi_r$. Všechny epipolární linie jedné kamery se setkávají v jednom bodě (epipólu).




Obrázek 3.3: Epipolární geometrie - pohled ze dvou kamer[1].

Až po procesu kalibrace[2] je vhodné přistoupit k samotné **rektifikaci**. To je proces, kdy se najdou korespondence mezi body v obou obrazech, z těch se určí homografie (vzájemná transformace mezi snímky) a na závěr transformujeme snímky tak, aby se korespondující body mezi oběma snímky nacházely na stejných vertikálních souřadnicích [20]. Názorný příklad vidíme na obrázku 3.4. Takto upravené snímky z kamer nám dále velmi pomohou při




Obrázek 3.4: Rektifikace snímků - první řádek znázornuje původní snímky z kamer, druhý řádek pak výsledek rektifikace.

[1]Obrázek z http://www.fbmi.cvut.cz/files/predmety/3528/public/Stereo%20vid%C4%9Bn%C3%AD%20-%20podklady.pdf

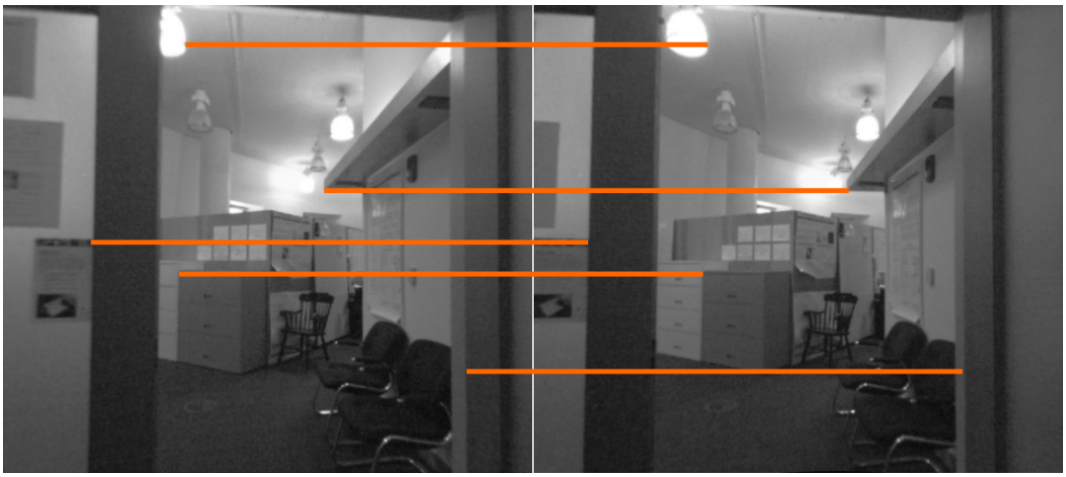
[2]Rektifikaci lze provádět i bez kalibrace kamery, nicméně tím přijdeme o přesnost a pro budoucí zpracování snímky upravené kalibračnímy parametry potřebujeme.

výpočtu disparitní mapy a obecně při hledání korespondencí mezi oběma snímky. Spousta algoritmů totiž používá hledání korespondenčních bodů a pokud již máme snímky jednou rektifikované, stačí nám pro nalezení korespondencí prohledávat pouze 1D prostor - tedy řádek, který je na totožné vertikální souřadnici jako vzorový bod. Zatímco u původních (nerektifikovaných) snímků bychom museli prohledávat snímek celý (tedy dvě dimenze).

### 3.2.2 Výpočet disparitní mapy

Disparitní mapa v sobě uchovává prostorovou (hloubkovou) informaci o scéně. Výhodou je, že jsme schopni tuto informaci získat z přímých dat kamery, takže se vyhneme zbytečným transformacím, které zavisí na odhadovaných parametrech. Ještě před samotným výpočtem disparitní mapy je vhodné částečně odstranit z obrazu šum. Pro předzpracování tedy použijeme některý z obrazových filtrů.

Poté můžeme přistoupit k samotné disparitě. Ta nám v základním smyslu vyjadřuje rozdíl. Konkrétně rozdíl mezi pozicemi stejného bodu na dvou různých snímcích scény - v případě stereo kamery tedy na levém a na pravém snímku. Čím blíže je objekt ke kameře, tím je jeho „skok“ mezi snímky větší. Princip výpočtu disparity je tedy založen na **rozdílu korespondujících si pixelů** mezi levým a pravým snímkem, viz následující obrázek 3.5. Na obrázku můžeme vidět, že vždy uvnitř jednoho páru korespondenčních bodů mají oba



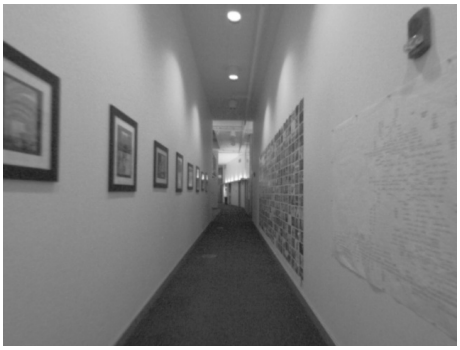
Obrázek 3.5: Ukázka korespondence pixelů v levém a pravém snímku.

body stejnou vertikální souřadnici. To je díky tomu, že máme snímky již rektifikované, viz sekce 3.2.1. Disparita se pak spočítá jako:

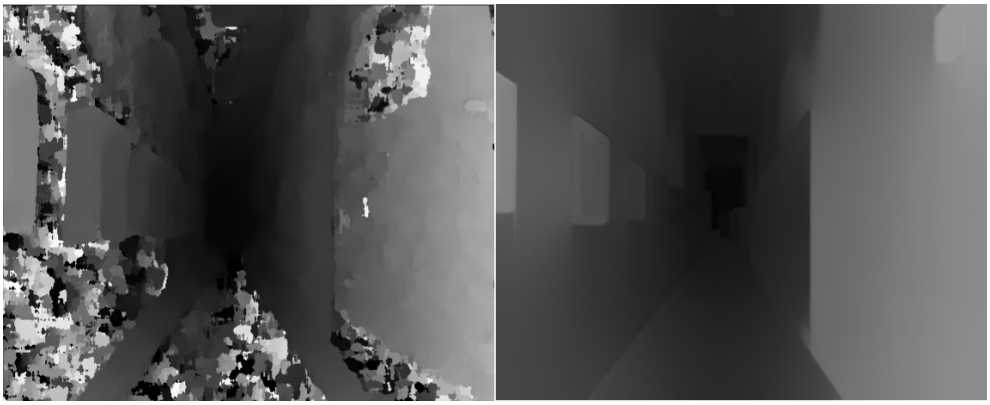
$$d = (x_l - x_{cl}) - (x_r - x_{cr}), \tag{3.3}$$

kde $[x_l, y_l]$ a $[x_r, y_r]$ jsou odpovídající si body (koncové body úseček na obrázku 3.5) levého a pravého snímku. Souřadnice $y$ je v párech stejná a $x_{cl}$ a $x_{cr}$ jsou indexy středu levého a pravého snímku.

Vzhledem k tomu, že po výpočtu disparitní mapy se nám znepřesní některé informace (například přijdeme o ostré hrany v obraze) a je potřeba upravit i homogenní oblasti, doporučujeme použít další filtr, například založený na metodě nejmenších čtverců - WLS (Weighted Least Squares) [23]. Ten nejenže obraz vyhladí, ale také obnoví hrany vzhledem k původnímu snímku. Výsledek aplikace takového filtru vidíme na obrázku 3.7.



Obrázek 3.6: Zdrojový obrázek pro následující ukázky.



Obrázek 3.7: Vlevo - přímá mapa disparity, vpravo - následné zpracování filtrem. Světlé body reprezentují velkou disparitu, naopak tmavé body malou.

Z disparitní mapy pak můžeme zpětně zrekonstruovat ortogonální 3D scénu. Přesněji se jedná o **zpětnou perspektivní projekci**. Pro takový výpočet ale už potřebujeme znát některé odhadnuté parametry například z kalibrace kamery. Pixely pak přepočítáme následovně:

$$z = \frac{fB}{d}, \tag{3.4}$$

$$x = \frac{(u - u_c)z}{f}, \tag{3.5}$$

$$y = \frac{(v - v_c)z}{f}, \tag{3.6}$$

kde $f$ je ohnisková vzdálenost, $B$ je báze (baseline), $[u, v]$ je pozice počítaného pixelu (index) v disparitní mapě a $[u_c, v_c]$ je pozice pixelu uprostřed snímku. Pro více detailů můžete nahlédnout do [1, 42]. My však tento krok necháme úplně na závěr našeho přístupu kvůli důvodům zmíněným v sekci 3.1, viz také naše schéma postupu 3.2.

Všimněme si ale, že podle principu výpočtu disparity nastávají problémy s **homogenními regiony**, kde je problém nalézt odpovídající si body mezi levým a pravým snímkem (viz obrázek 3.5 a 3.7 vlevo). V určitých situacích se s tímto problémem setkáváme i v naší

práci, ale jelikož tohle není cílem práce, moc jsme se jím nezabývali a spokojili jsme se s aktuální implementací dle dostupných knihoven. O tomto problému a jeho řešení se ale můžete dočíst například v [17].

### 3.2.3 Detekce pozemní roviny

V disparitní mapě se pokusíme najít pozemní rovinu, po které se robot pohybuje. Jak jsem již zmínila v 3.1, máme více způsobů, jak pozemní rovinu před robotem detekovat. Například pomocí histogramové metody, tzv. v-disparity, jejíž nevýhody jsme si popsali výše. Dále bychom mohli předpokládát, že bude mít rovina neustále konstantní výšku. To se hodí maximálně pro roboty, které jezdí uvnitř budov, kde se výška roviny vůči kameře příliš nemění. Ale ani tak to není dostatečně spolehlivé. V nejvíce případech bude fungovat **dynamický přístup**.

V naší metodě jsme k tomu použili **geometrický model roviny** definovaný následovně:

$$ax + by + cz - d = 0, \tag{3.7}$$

respektive

$$ax + by + cz = d, \tag{3.8}$$

kde $n = (a, b, c)$ je normálový vektor roviny. Pak vzdálenost bodu $i$ (souřadnice bodu $i$ jsou $[x_i, y_i, z_i]$)od roviny (3.8) je vypočítána jako

$$R_i = ax_i + by_i + cz_i - d \tag{3.9}$$

za předpokladu, že je rovina normalizovaná a platí tedy

$$a^2 + b^2 + c^2 = 1. \tag{3.10}$$

Pokud rovinu nenormalizujeme, zůstáváme u výpočtu

$$R_i = \frac{ax_i + by_i + cz_i - d}{\sqrt{a^2 + b^2 + c^2}}. \tag{3.11}$$

#### Odhad roviny metodou RANSAC

Výše jsme si tedy definovali náš použitý model roviny. Pro získání konkrétní roviny nyní potřebujeme například 3 body pro. V některých případech může být rovina definována i přímkou a jedním bodem, ale my se držíme minimálních prostředků pro definici roviny, pro kterou tedy potřebujeme právě 3 body. Za tímto učelem jsme zvolili velmi rychlou a efektivní metodu vyvinutou komunitou počítačového vidění - **Random Sample Consensus (RANSAC)**. Postup této metody v našem případě je následující:

1. náhodně vybereme 3 vzorky (body) v oblasti zájmu (ROI - region of interest) v disparitní mapě pro hrubý odhad roviny,
2. těmito body proložíme rovinu dle modelu 3.8 a určíme pro každý bod $P_a$ v ROI vzdálenost od této roviny v disparitní mapě, a to následovně:

$$MIN\_DISTANCE = (P_a - P_b)\frac{n}{\|n\|}, \tag{3.12}$$

kde $n$ je normálový vektor roviny a $P_b = (u_b, v_b, d_b)$ je bod v rovině, kde

$$au_b + bv_b - 1d_b + c = 0. \tag{3.13}$$

Do vzdálenosti započítáme drobnou odchylku a rozhodneme, zda jsou jednotlivé body tzv. **inliers** - splňují definici roviny s nějakou odchylnou, či **outliers** - nesplňují definici roviny,

3. výše zmíněné kroky budeme opakovat, dokud platí podmínka

$$N > sample\_count, \tag{3.14}$$

kde $sample\_count$ je počet pokusů (vzorků) roviny pro nalezení dostatečně přesné roviny - každým průchodem cyklu se o 1 navýší a

$$N = \frac{log(1-p)}{log(1-(1-e)^s)}, \tag{3.15}$$

kde $s$ je počet vzorků na proložení modelu (v našem případě 3 pro rovinu), $p$ je pravděpodobnost nastavena na hodnotu $0,99$ a

$$e = 1 - \frac{number\_of\_inliers}{number\_of\_points} \tag{3.16}$$

v aktuálním cyklu smyčky.

Tímto algoritmem tedy získáme **hrubý odhad roviny** z disparitní mapy, po které se robot pohybuje. Bližší detaily ohledně metody RANSAC můžete shlédnout v [11].

### Zpřesnění roviny metodou nejmenších čtverců

Pro upřesnění polohy a náklonu roviny použijeme **metodu nejmenších čtverců**[3]. Máme tedy body (inliers) které nám modelují aktuální nejlépe získanou rovinu pomocí metody RANSAC. Tuto množinu bodů si označíme $N$, přičemž každý bod $i$ z této množiny má souřadnice $[x_i, y_i, z_i]$. Z principu metody nejmenších čtverců chceme minimalizovat $Q$:

$$Q = \sum_{n=1}^{N} R_i^2, \tag{3.17}$$

kde $N$ je již zmiňovaná množina bodů inliers a $R_i$ je vzdálenost bodu od roviny z (3.11). Můžeme tedy rovnici rozepsat jako:

$$Q = \sum_{n=1}^{N} (ax_i + by_i + cz_i - d)^2. \tag{3.18}$$

Jako další krok minimalizace potřebujeme získat parciální derivace, tedy:

$$\frac{\partial Q}{\partial a} = \sum_{n=1}^{N} 2x_i(ax_i + by_i + cz_i - d) = 0, \tag{3.19}$$

[3] Least Square Plane (LSQP)

$$\frac{\partial Q}{\partial b} = \sum_{n=1}^{N} 2y_i(ax_i + by_i + cz_i - d) = 0, \tag{3.20}$$

$$\frac{\partial Q}{\partial c} = \sum_{n=1}^{N} 2z_i(ax_i + by_i + cz_i - d) = 0, \tag{3.21}$$

$$\frac{\partial Q}{\partial d} = \sum_{n=1}^{N} -2(ax_i + by_i + cz_i - d) = 0. \tag{3.22}$$

Všimněme si v (3.22), že platí

$$\sum_{n=1}^{N} d = Nd. \tag{3.23}$$

Potom můžeme rovnici (3.22) přepsat jako:

$$d = ax_0 + by_0 + cz_0, \tag{3.24}$$

kde

$$x_0 = \frac{\sum_{n=1}^{N} x_i}{N}, \qquad y_0 = \frac{\sum_{n=1}^{N} y_i}{N}, \qquad z_0 = \frac{\sum_{n=1}^{N} z_i}{N}. \tag{3.25}$$

Rovnice (3.24) ukazuje, že nejlepší rovina prochází těžištěm. Odečtením tohoto těžiště od každého bodu a dosazením do rovnic (3.19 - 3.21) dostaneme rovnice, které můžeme obecně zapsat jako

$$WP = 0, \tag{3.26}$$

kde

$$W = \begin{bmatrix} \sum_{n=1}^{N} (x_i - x_0^2) & \sum_{n=1}^{N} (x_i - x_0)(y_i - y_0) & \sum_{n=1}^{N} (x_i - x_0)(z_i - z_0) \\ \sum_{n=1}^{N} (x_i - x_0)(y_i - y_0) & \sum_{n=1}^{N} (y_i - y_0)^2 & \sum_{n=1}^{N} (y_i - y_0)(z_i - z_0) \\ \sum_{n=1}^{N} (x_i - x_0)(z_i - z_0) & \sum_{n=1}^{N} (y_i - y_0)(z_i - z_0) & \sum_{n=1}^{N} (z_i - z_0)^2 \end{bmatrix} \tag{3.27}$$

a

$$P = \begin{bmatrix} a \\ b \\ c \end{bmatrix}. \tag{3.28}$$

Pro vyřešení metody nejmenších čtverců nám stačí vyřešit rovnici (3.26). Musíme se ale vyhnout triviálnímu řešení kdy $a = b = c = 0$. Proto zavedeme podmínku pro koeficienty roviny, a to nejlépe

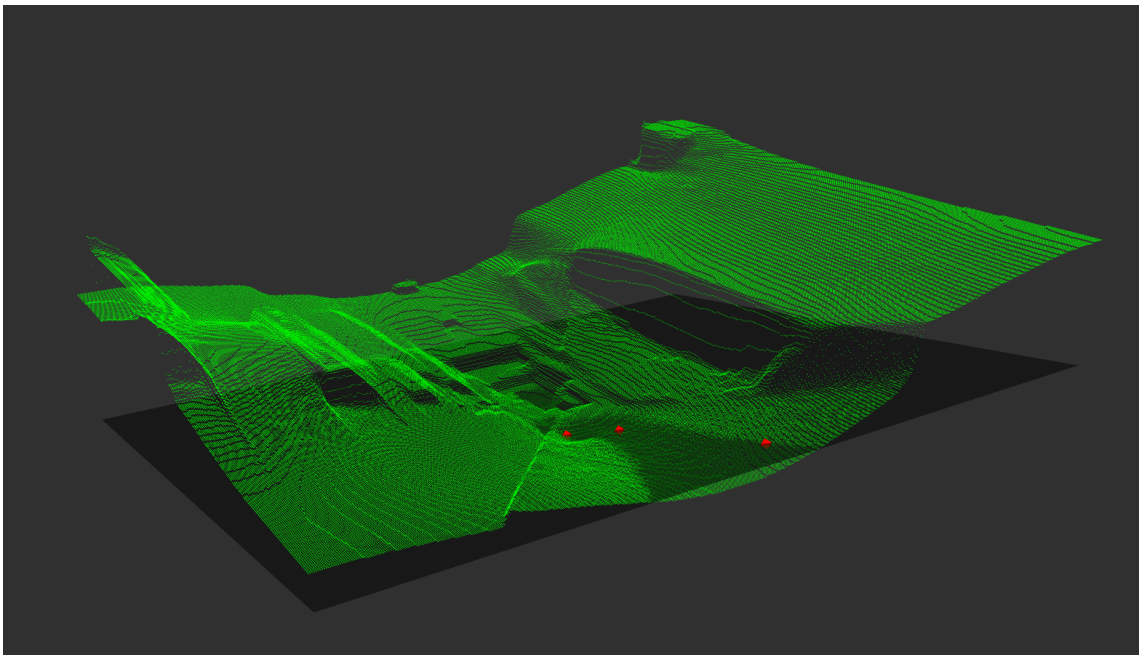
$$a^2 + b^2 + c^2 = 1. \tag{3.29}$$

Podmínka je zároveň velmi vhodná, protože se nám hodí také pro normalizaci roviny, viz (3.10) zmíněná výše. Dále s touto podmínkou a rovnicí (3.26) dostáváme tzv. **problém vlastních hodnot** (Eigenvalue problem). Tím se dostáváme k poslední rovnici:

$$WE = VE, \tag{3.30}$$

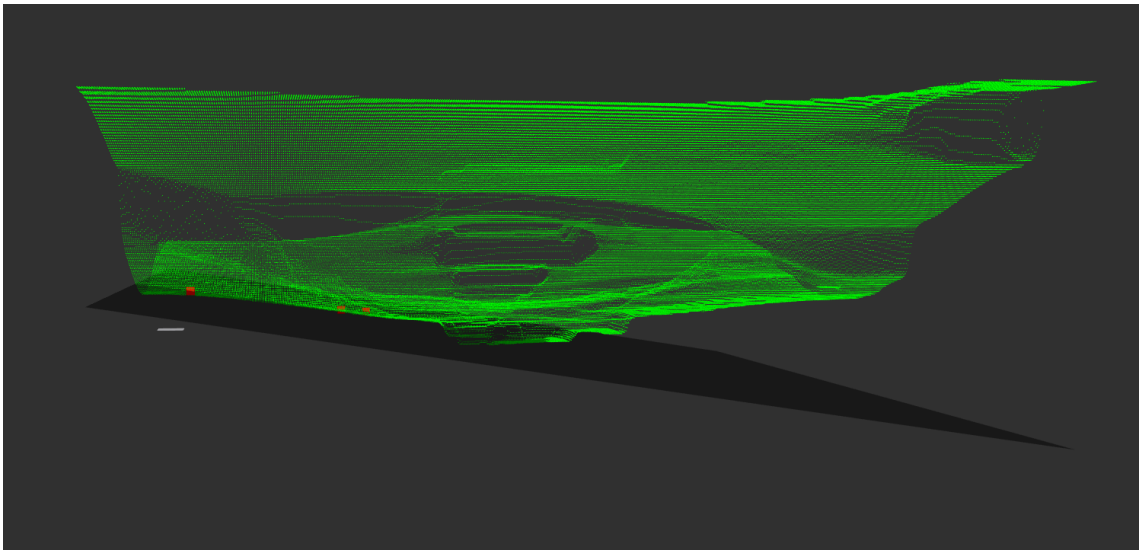
kde $V$ je vektor vlastních hodnot a $E$ je vektor vlastních vektorů. Tato rovnice nám vrátí 3 vlastní hodnoty a matici o velikosti 3x3 (vlastní vektory). Hodnoty $V$ tedy reprezentují námi hledaný součet vzdáleností nejmenších čtverců a vektory v $E$ nám vrací 3 (po řádcích) odpovídající nastavení parametrů roviny, tedy hodnoty $a$, $b$, a $c$. Řešením se pak stává minimální hodnota z $V$ a odpovídající vektor z $E$, ze kterých dostáváme jednoznačnou definici roviny získanou metodou nejmenších čtverců. Je užitečné dále vědět, že všechny 3 vektory ve vektoru $E$ jsou vzájemně ortogonální. Reprezentují tedy nejlepší, nejhorší a nějaké střední řešení (rovinu). Dále pokud jsou vlastní hodnoty příliš podobné až stejné, znamená to že jsou vzorky (body) špatné - symetrické. Náhledy na přesnější vysvětlení či podrobnější výpočty celého postupu jsou k dispozici v [27, 31, 4].

Výsledky metody RANSAC následované metodou nejmenších čtverců aplikované na zdrojový snímek 3.6 vidíme na následujících obrázcích:



Obrázek 3.8: Detekovaná rovina v disparitní mapě ze snímku 3.6.

Zelené body v obrázku reprezentují disparitu zobrazenou v prostoru. Červené body (3) definují rovinu nalezenou metodou RANSAC a metodou nejmenších čtverců. Těmito body je následně proložena černá rovina přes celou oblast velikosti snímku, která byla našim přístupem detekována. Na obrázku vidíme, že detekce roviny velmi záleží na kvalitě disparitní mapy. Zrovna v tomto případě se může stát, že bude mít výsledná rovina špatný sklon. Dále takovýto výsledek bude mírně omezovat následující krok - určování překážek. Na dalším obrázku 3.9 vidíme detailněji správný náklon pozemní roviny roviny směrem od dolního okraje disparitní mapy do středu snímku.



Obrázek 3.9: Detekovaná rovina v disparitní mapě při pohledu z boku (zprava) pro lepší ukázku náklonu roviny.

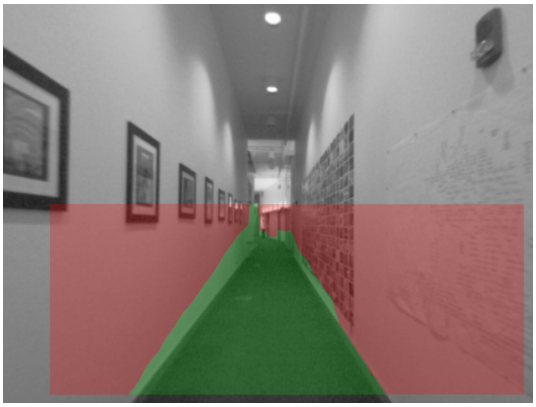
### 3.2.4 Určení překážek

Nacházíme se v rozhodovací fázi, kde máme za úkol určit, které body v disparitní mapě představují pro robota **překážku**, a které naopak znamenají **volný průchozí prostor**. Za tímto účelem použijeme jednoduchý algoritmus. Jelikož máme rovinu, po které se robot pohybuje, stačí nám určit dvě konstanty, spodní a horní limit, které budou znázorňovat vzdálenost v disparitní mapě právě od naší nalezené roviny. Formálně to zapíšeme jako:

$$p_d(x,y) = \begin{cases} free, \Delta d < low \\ occupied, low \leq \Delta d \leq high \\ free, \Delta d > high \end{cases} \tag{3.31}$$

kde $p_d(x,y)$ je pixel v disparitní mapě, $\Delta d$ je rozdíl mezi očekávanou disparitou (ležící na rovině) a naměřenou disparitou a $low$ a $high$ jsou dva prahy, které jsme získaly z experimentů na našich datech.

Obrázek 3.10 ukazuje, jak vypadá výsledek takového rozhodování. Červené pixely znamenají obsazenost, zelená pak volnou průjezdnou plochu pro robota. Na obrázku je vidět, že spodní části překážek náš přístup označil částečně jako volné plochy. To je způsobeno tím, že pro body ležící v rovině musíme povolit odchylku (pro vzdálenost bodu od nalezené roviny), která je také zavislá na přesnosti disparitní mapy. A jelikož disparitní mapa k tomuto snímku vypadá jako na obrázku 3.8, kde je vidět značné prohloubení v prostoru pozemní roviny, pro zařazení všech bodů v této rovině bylo potřeba odchylku poděkud povolit. Takto mylně označené oblasti v dolních částech překážek nám ale ve výsledku ničemu nevadí, protože po promítnutí překážek do 2D mapy obsazenosti překryjí tuto dolní část právě vyšší části překážek. To si ukážeme v následující kapitole 3.2.6.

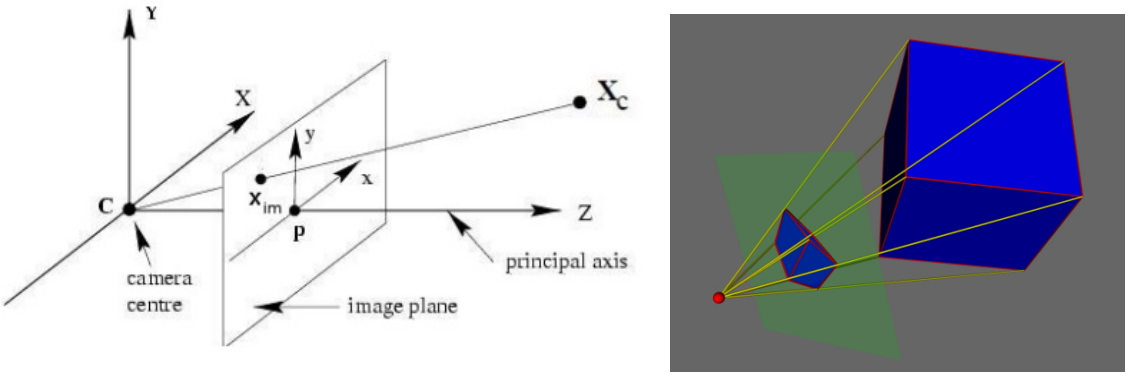


Obrázek 3.10: Výsledek rozhodování mezi volnou a obsazenou plochou.

### 3.2.5 Zpětná perspektivní projekce

Až doposud jsme pracovali s přímými daty z kamery. To je jedna z hlavních výhod našeho přístupu. Až těsně před fází zápisu získaných informací do mapy budeme muset data transformovat. Tento krok jsme mohli podstoupit i dříve, například hned při získání disparitní mapy. Pro tuto transformaci jsou ale potřeba odhadnuté parametry z kamery, kderé nám do dalších výpočtů zanáší chybu. Proto jsme se rozhodli tuto fázi přesunout až na samotný konec celého řetězce, viz 3.2.

Základem všech našich výpočtů byla tedy disparitní mapa. Ta vznikla z obou kamer, přičemž oba snímky byly vždy zkresleny perspektivní projekcí, viz ukázka 3.11.




Obrázek 3.11: Princip perspektivní projekce[4] - vlevo, ukázka deformace na promítací rovině[5] - vpravo.

Princip je takový, že veškeré body v prostoru (v levém obrázku například bod $X_C$) jsou promítány na průnik paprsku, který vychází z kamery (jedná se o polopřímku $\overrightarrow{CX_C}$, kde bod $C$ je camera centre), s **projekční rovinou** kamery (image plane). Výsledný bod se pak ze 3D prostoru promítne na tuto 2D rovinu jako $x_{im}$. Když promítneme několik takových bodů, respektive celý objekt, dostáváme něco podobného jako na obrázku 3.11 vpravo. Červený bod vlevo je opět střed promítání a zelená plocha znázorňuje promítací

[4]Obrázek z https://gamedev.stackexchange.com/questions/61406/what-is-the-logic-behind-a-3d-projection-camera-perspective

[5]Obrázek z http://uivty.cs.cas.cz/~horcik/Teaching/applications/node4.html

rovinu. Zde již vidíme, že 3D objekt (modrá krychle) nezachovává po promítnutí ze 3D do 2D správné **poměry stran ani rovnoběžnosti**. A přesně takto vypadá naše disparitní mapa.

Robot si ale při mapování prostředí představuje prostor ve 3D ortogonálním systému souřadnic - tedy přesně takový systém, v jakém je vyobrazena původní modrá krychle v prostoru na obrázku 3.11 vpravo. V našem případě, když se podíváme na mapy vytvářené LIDAR senzorem (obrázky 2.17 a 2.10), také vidíme, že mapy jsou v ortogonálním systému souřadnic (ikdyž jen ve 2D), neboť zachovávají rovnoběžnost (přímé cesty na snímcích). Jako důkaz, že jsou tyto mapy z laseru ortogonální, stačí princip snímání laserovým sensorem, viz sekce 2.2.1. Tato data jsou totiž přímo vyobrazena do mapy bez jakékoliv transformace. Data z kamery chceme také zakreslit do 2D mapy obsazenosti, viz další kapitola 3.2.6. A právě proto musíme na získané detekované překážky, které chceme do této mapy zakreslit, aplikovat **zpětnou perspektivní projekci** z 2D projekční roviny zpět do 3D ortogonálního prostoru. Pro tyto potřeby nám poslouží příslušná transformační matice [32], ve které budou použity parametry získané z kalibrace kamery (viz sekce 3.2.1), nebo potom přímým přepočtem zmíněným v rovnicích 3.4, 3.5 a 3.6.

### 3.2.6 2D mapa obsazenosti z kamery

Na vstup tohoto závěrečného bloku celého řetezce našeho přístupu dostáváme transformované body (tzv. **point cloud**) v ortogonálním 3D systému, u kterých víme, zda jsou překážkou či nikoliv. Jelikož budeme z těchto bodů sestavovat pouze 2D mapu, zbavíme se třetí souřadnice následovně. Víme, které body nám reprezentují překážky a které ne - ty které jsou překážkami, pouze ty budeme chtít do mapy zakreslit a s těmi tedy budeme pracovat. Jelikož máme u všech těchto bodů souřadnice $[x, y, z]$ kde jsou jednotlivé souřadnice reprezentované v ortogonálním systému kamery (viz souřadnicový systém kamery 3.11 vlevo), stačí nám pouze vynechat souřadnici $y$, protože nás už dále výšková informace nebude zajímat. Tím tedy promítneme bod ze 3D prostoru do požadovaného 2D prostoru rovnoběžného s LIDARem.

V našem přístupu jsme se rozhodli do mapy z kamery zakreslovat pouze překážky (**stav obsazeno**), zbytek prostředí zůstane v **neznámém stavu** (oproti LIDARu, který má ve své mapě stavy 3). Mapa bude tedy pravidelná mřížka $M$. V každé buňce uchovává hodnotu v intervalu $[0, 1]$, která indikuje, zda je buňka obsazena či nikoliv. Vysoká hodnota znamená, že je buňka obsazena a obráceně. My však náš případ pro kameru zjednodušíme a buňka bude obsazena pouze pokud má hodnotu vyšší než $0,5$ a v opačném případě přepneme buňku do druhého stavu, tedy stav neznámý. Buňky jsou na sobě nezávislé a cílem tedy je odhadnout pravděpodobnost obsazenosti

$$P(M_i|x_{1:t}, z_{1:t}) \tag{3.32}$$

pro každou buňku v mřížce $M_i$, během několika pozorování kamerou $z_{1:t} = \{z_1, ..., z_t\}$ na odpovídajících pozicích $x_{1:t} = \{x_1, ..., x_t\}$.

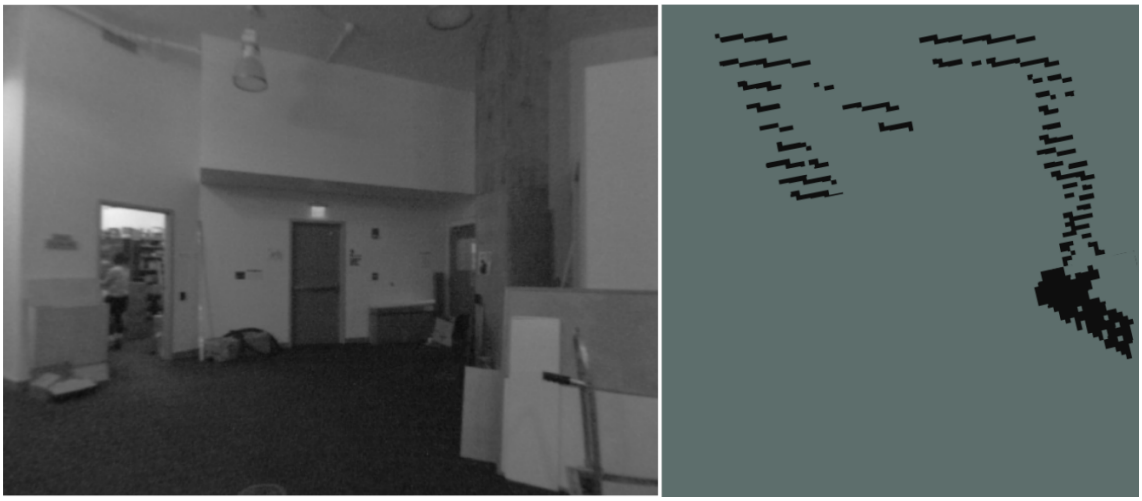
Existuje několik metod, jak vypočítat vždy aktuální hodnotu v mřížce. My pro tento případ použili schéma **Bayesova aktualizačního pravidla** z [36], které nám poskytuje elegantní rekurzivní zápis pravděpodobnosti v tzv. log-odds formě:

$$logO(M_i|x_{1:t}, z_{1:t}) = logO(M_i|x_{1:t-1}, z_{1:t-1}) + logO(M_i|x_t, z_t) - logO(M_i). \tag{3.33}$$

kde

$$O(a|b) = odds(a|b) = \frac{P(a|b)}{1 - P(a|b)}. \tag{3.34}$$

V 3.33 je $P(M_i)$ počáteční nastavená pravděpodobnost, která je nastavena právě na hodnotu $0,5$, což už nám reprezentuje neznámý stav. Tím nám tato komponeta ze vzorce jakoby zmizí. Zbylá pravděpodobnost, tedy $P(M_i|x_t, z_t)$, je nazývána **modelem inverzního senzoru**. Specifikuje pravděpodobnost buňky $M_i$ podle právě aktuálního jednoho měření $z_t$ z místa $x_t$. Díky této rekurzivní reprezentaci algoritmu, jak aktualizovat mřížku obsazenosti, můžeme provést jednoduchou inkrementální implementaci problému po každém příchodu vzorku dat. Detaily tohoto přístupu můžete shlédnout v [38]. Výsledek naší mapy z kamery vidíme na obrázku 3.12. Jak jsem již zmínila, černé buňky reprezentují překážky a modrozelené buňky reprezentují neznámý stav.



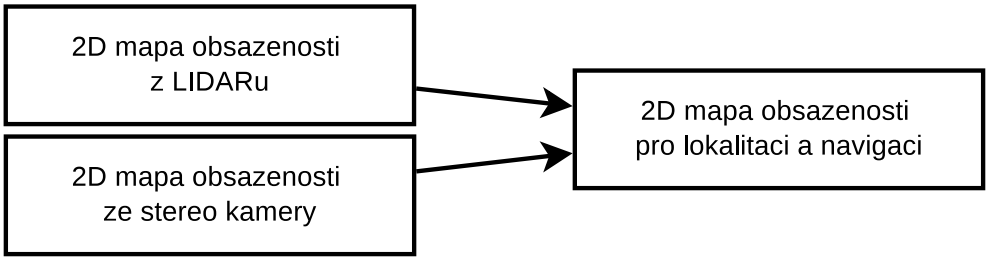
Obrázek 3.12: 2D mapa obsazenosti získaná z kamery (vpravo), zdrojový snímek vlevo.

Záměrně se vyhýbáme určování volných (bílých) buněk, protože je kamera vůči LIDARu nepřesná. Proto toto důležité rozhodování necháváme na LIDARu, které bude popsáno v následující kapitole.

### 3.2.7 Fúze map z LIDARu a kamery

Toto je poslední fáze našeho nového přístupu. Na vstupu máme 2D mapu obsazenosti z LIDARu a 2D mapu obsazenosti ze stereo kamery, kterou jsme získali implementací našeho přístupu popsaného výše. Z kamery jsme si dovolili udělat 2D mapu proto, že se robot pohybuje pouze ve dvou osách, takže třetí rozměr nepotřebujeme (za předpokladu, že se ve 2D mapě promítnou výškové informace o překážkách, které pohyb robota omezují). Dále stavíme na tom, že LIDAR je přesnější a spolehlivější senzor, proto jsme se rozhodli za základní mapu považovat mapu z LIDARu a do té přidáme informace z kamery. Výsledná mapa nám pak může pomoci v lokalizaci nebo navigaci robota. Schéma vstupů a výstupu fúze je vyobrazeno na 3.13.

Nicméně, jak si v jedné z dalších kapitol ověříme experimenty, pro lokalizaci se tento přístup velmi neosvědčil. Je to především kvůli tomu, že ve valné většině případů se laserový senzor má čeho (překážek a dalších objektů) chytit. A je to dost přesný senzor, na který se




Obrázek 3.13: Vstupy a výstup závěrečné fúze.

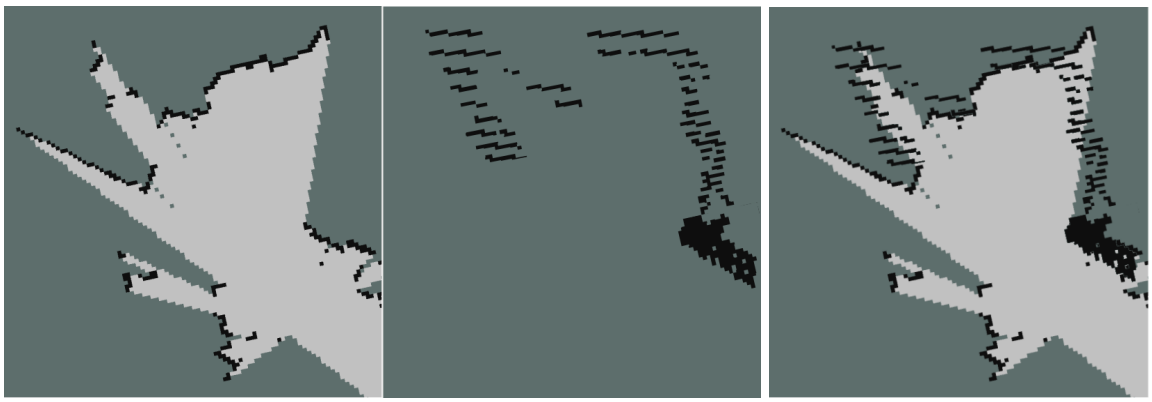
dá při lokalizaci (metodě SLAM) spolehnout. Kamera je oproti laserovému senzoru relativně nepřesná. Zanášela by do lokalizace a mapování pouze větší chybu. V menšině případů, kdy se LIDAR nemá čeho chytit, nebo nastane výpadek tohoto senzoru, se dá na krátkou dobu na kameru spolehnout. Je však potřeba zde pohlídat správné přepínání priorit (hodnot pravděpodobností) v závislosti na prostředí, ve kterém se robot pohybuje.

Mnohem lepší využití však náš přístup nachází v navigaci mobilních robotů (detaily budou probrané v experimentech a závěru). Proto jsme však celou část lokalizace a mapování ponechali na laserovém senzoru (LIDARu) a fúzovanou mapu budeme používat výhradně pro navigaci robota.

Pro tyto potřeby jsme aplikovali jednoduchou, ale účelnou fúzi. Obsazené buňky z mapy kamery jsou mapovány do mapy LIDARu následovně:

$$M_i = \begin{cases} free, (L_i = free) \wedge (C_i = unknown) \\ unknown, (L_i = unknown) \wedge (C_i = unknown) \\ occupied, (L_i = occupied) \vee (C_i = occupied) \end{cases} \tag{3.35}$$

kde $M_i$ je buňka ve výsledné fúzované mapě $M$ na indexu $i$, $L_i$ je buňka z mapy LIDARu na indexu $i$ a konečně $C_i$ je buňka z mapy stereo kamery na indexu $i$. Výsledek takovéto fúze může vypadat následovně: První (levá) mapa je z dat LIDARu. Druhá (prostřední) je



Obrázek 3.14: Ukázka fúze obou map ze zdrojového snímku 3.12 vlevo.

ze vytvořena ze stereo kamery. Třetí (vpravo) je výsledná fúze obou map. Na první pohled je znát jednoduchost spojení map, která spočívá v podobnosti logicé operace $OR$. Pro naše účely práce nám takovéto spojení stačí. I zde se dá ale leccos zlepšit. Pak také záleží na rozhraní, které bude mezi plánovačem trasy (včetně výpočetního algoritmu plánovače) a modulem SLAM.

# Kapitola 4

# Implementace

## 4.1 Knihovna OpenCV

OpenCV (Open Source Computer Vision) je knihovna soustředěná především na zpracování obrazu a to v reálném čase. Obsahuje širokou bázi různých implementovaných algoritmů, výbornou dokumentaci a hlavně jsou tyto algoritmy / funkce dostatečně otestované a efektivně napsané. Dnes se už řadí mezi multiplatformní a volné knihovny pod BSD licencí. Má podporu rozhraní nejen pro jazyky C/C++ (ve kterých je knihovna optimalizována), ale také pro jazyky Python a Java. K tomu všemu má oficiálně komunitu větší než 47 tisíc uživatelů a počet stažení překročil hranici 14 milionů. Oblasti použití jsou od interaktivního umění až po pokročilou robotiku. Jmenujme jich několik konkrétně:

- rozpoznávání gest,
- mobilní robotika,
- segmentace obrazu,
- rozšířená realita,
- neuronové sítě a další...

Veškerá dokumentace této knihovny je dostupná online[1]. Existuje i kvalitní tištěná literatura [5].

## 4.2 Robotický operační systém (ROS)

Robot Operating System, zkráceně ROS, se stal především v oblasti mobilní robotiky velice populárním a používaným frameworkem. Používají ho tisíce lidí na celém světě. V roce 2013 bylo oficiálních uživatelů přes 1500, přes 3000 lidí pracujících na online dokumentaci a 5700 lidí poskytujících online podporu.

Systém je šířen pod licencí BSD - je tedy distribuován jako otevřený, volně šiřitelný software a je zdarma. Plně podporován je ROS pro operační systém Linux Ubuntu, experimentální provoz je však na mnoha dalších platformách. Velkou výhodou ROSu je podpora mnoha senzorů, motorů a celých robotů. Aktuálně podporuje asi 45 komerčních robotů.

[1] Na adrese http://opencv.org/

### 4.2.1 Koncept

ROS je navržen tak, aby byl co nejvíce modulární a flexibilní. Uživatelům tedy umožňuje použít jen jeho některé části a některé části si pak může uživatel vytvořit sám. Výsledný program se skládá z jednotlivých procesů neboli uzlů. Uzly spolu vzájemně komunikují prostřednictvím zpráv. Díky rozdělení do uzlů je možné používat jeden kód pro více aplikací, případně spouštět uzly na různých počítačích.

Systém je rozdělen na **dvě základní vrstvy:** jádro, které je spravováno kalifornskou společností Willow Garage a robotický software, který je dílem ROS komunity.

### 4.2.2 Služby a nástroje

ROS také poskytuje širokou škálu služeb:

- Propojení různého softwaru na různých platformách
- Zasílání zpráv mezi procesy (uzly)
- Logování (zpětná efektivní detekce problémů)
- Parametr server (sdílený prostor, ve kterém lze uchovávat parametry - tyto lze zpětně načíst z libovolného uzlu)
- Vzdálené volání procedur
- Nástroje (RViz, Gazebo, RQt)
- Knihovny (OpenCV, Qt)

Mezi poskytované služby patří (některé již zmíněné) **grafické nástroje**, jako například: Stage (2D simulátor), Gazebo (3D simulátor), RViz (3D vizualizační systém), RQt (framework pro grafické uživatelské rozhraní) a OpenCV (zpracování obrazu). Ty nejdůležitější rozebereme podrobněji. Detaily ohledně ROSu můžete shlédnout online [2].

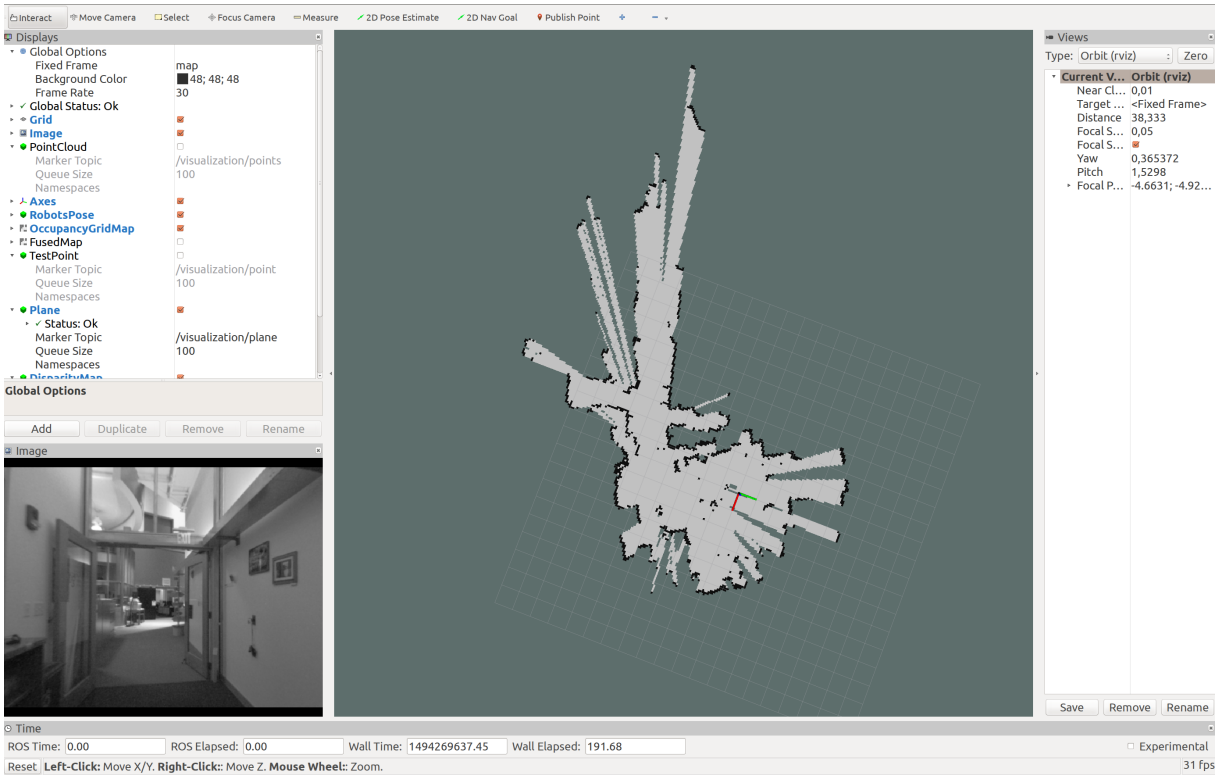
### 4.2.3 RViz

RViz je **3D vizualizační nástroj**, který umožňuje sledovat: co robot dělá, co vidí a také jak přemýšlí. Díky **systému transformací** můžeme sledovat robotův svět z několika úhlů pohledu (jeden souřadný systém bude fixní a ostatní se do něj transformují). Nástroj má možnost zobrazit moduly před tzv. **Docked window** (reálný pohled z kamery v jednom okně, geometrické elementy z pohledu robota v dalších oknech). Uživatelské rozhraní tohoto nástroje pak poskytuje uživateli několik **měření a náhledů** při sledování robota: po kliknutí do vizualizovaného prostoru získáme souřadnice daného bodu, můžeme měřit reálné vzálenosti za použití pravítka. Dále máme možnost výběru a pohybu s objekty a nastavení úhlu pohledu v podstatě odkudkoli s tím, že vždy bude jeden systém souřadnic fixní, tzv. **fixed frame** a ostatní se vůči němu transformují.

Vstupní data do tohoto nástroje jsou různé objekty (vektory, úsečky, body, krychle, válec) reprezentované převážně pomocí tzv. **Markers**, kterým řekneme pozici a souřadný systém, do kterého chceme objekt zakreslit, barvu a čas. Dále se dají vizualizovat přímo některá data ze senzorů - například lazerové skeny či snímky z kamery. Dále umí RViz vizualizovat některé typické struktury z robotiky, jako je například právě naše používaná

[2] Na adrese http://www.ros.org/

**Occupancy Grid Map**. Výstup v RVizu pak vypadá například jako na obrázku 4.1. Levá horní část obsahuje nastavení, co chceme všechno zobrazit v hlavní části uprostřed, kde máme aktuálně mapu obsazenosti. V levé dolní části pak vidíme zvlášť okno s pohledem z kamery a v pravé části máme pak dále nastaveny detaily pohledu a pozice, ze které scénu sledujeme. Více o RVizu se můžete dozvědět online [3].




Obrázek 4.1: Grafické uživatelské rozhraní RVizu s ukázkou dat a s přidaným oknem pohledu kamery.

### 4.2.4 Gazebo

Gazebo je robotický 3D grafický simulátor, pomocí kterého je možné simulovat aplikaci v plné míře. Propojení s ROSem je vytvořeno prostřednictvím zásuvných modulů. Díky tomu, že mají tyto uzly stejné rozhraní jako celý ROS, je možné vytvořit simulaci jako přímou náhradu za hardware. Vývoj začal v roce 2002 především jako simulátor pro venkovní prostředí. V dnešní době už ho ale většina uživatelů používá pro simulace i ve vnitřním prostředí. Od roku 2009 je Gazebo nejpoužívanější simulační nástroj v ROS komunitě.

**Výhody** tohoto simulátoru jsou následující. Simulátor nám umožňuje rychlé testování algoritmů, tvorbu regresivního testování za pomocí realistického scénáře, simulace chování robota ve složitých vnitřních i venkovních scénách. Od verze 1.9 není závislý na ROSu a lze ho instalovat jako samostatný balík. Dále obsahuje kvalitní zpracování fyzikálních zákonů (4 vysoce výkonné enginy: ODE, DART, Simbody a Bullet). Jednotlivé části jsou rozšiřitelné a objekty jsou popsány pomocí skriptů formátovaných v SDF.

[3] Na adrese http://wiki.ros.org/rviz/

Za **nevýhody** pak můžeme považovat jeho nízkourovňové programování. Dále fakt, že nepodporuje paralelizaci a nepodporuje zpracování simulace s vizualizací bez hardwarového displeje (nelze spouštět simulaci vzdáleně). A neřeší ani učení robotů a neposkytuje tedy ani informace, které nelze získat v reálném světě, ale byly by vhodné pro učení robota. Více se o tomto simulátoru můžete dočíst online[4].

## 4.3 IDE a cílová platforma

### Vývojové prostředí

Celou aplikaci jsem vytvářela ve vývojovém prostředí (Integrated Development Environment - IDE), které se nazývá **CLion**[5]. Jedná se o multiplatformní IDE pro jazyky C a C++. Poskytuje chytré formátování a napovídání. Díky modulu navigování se dostaneme snadněji k našim symbolům, třídám a souborům. Můžeme prozkoumávat volání funkcí a typovou hierarchii. Dále nám CLion poskytuje automatické generování nejpoužívanějších kódů, jako jsou například tzv. `get`tery a `set`tery a některé složitější šablony. K nápomoci nám může být také výborná analýza datového toku, která se nám snaží napovídat a navrhovat opravy. Mezi vestavěné nástroje pak patří klasická možnost sestavení, spuštění a debugování projektu. CLion podporuje tzn. unit testy (konkrétně Google testy) a poskytuje uživatelské rozhraní pro průzkum výsledků z těchto testů. Během vývoje se dá jednoduše sledovat i dokumentace kódu, která se dá externě zobrazit ve vyskakovacím okně. Celá dokumentace je postavena na Doxygen formátování. CLion také podporuje nejpopulárnější verzovací nástroje (VCS - Version Control System), jako jsou například Git, GitHub a Mercurial. Uvnitř CLionu je vestavěný terminál, díky kterému máme rychlý přístup na nízké urovni k našim programům.

Toto prostředí máme díky škole poskytované zdarma. Jednou z hlavních výhod a zároveň důvodů, proč jsem volila právě toto, je integrace sestavovacího systému (catkin) ROSu do tohoto prostředí. Stačí správně importoovat soubor `CMakeLists.txt` požadovaného modulu, se kterým chceme pracovat. Podobně lze importovat i `CMakeLists.txt` na nejvyšší úrovni (tedy nad celým projektem) a lze i tak jednotlivě pracovat s každým modulem zvlášť.

### Cílová platforma

Veškerý vývoj softwaru pro robora RUDA v mojí práci je vyvinut výhradně pro platformu **Linux**. Přesně se jedná o verzi **Ubuntu 16.04 64bit**, pro kterou právě nedávno vydali námi použitou nejnovější verzi systému ROS - **Kinetic**. Nevýhodou je, že zatím nejsou všechny balíčky v nejnovější verzi ROSu ještě nejsou zprovozněny a bývá potřeba udělat občas drobné úpravy v kódu, popřípadě doinstalovat některý balíček.

Tato nejnovější verze ROSu (a tedy i operačního systému Ubuntu 16.04, protože na nižší verzi Ubuntu se verze ROSu Kinetic jednoduše dát nedá) byla zvolena proto, že oproti starším verzím má některé nesporné výhody, které jsem sama v projektu zkusila. V prvním případě se jedná o přímou integraci knihovny OpenCV, která se dříve musela instalovat ručně a někdy pak docházelo k problémům s deklarací, když bylo předinstalováno více verzí OpenCV. Další důvod je, že nová verze ROSu nám odstraňuje výrazný bug v simulátoru Gazebo, kde za určitých (i nastavených fyzikálních) podmínek robotovi náhodně podkluzoval podvozek, ačkoli to bylo nežádoucí.

[4] Na adrese http://gazebosim.org/

[5] Dostupné na https://www.jetbrains.com/clion/

## 4.4 Zprovoznění senzorů

### 4.4.1 LIDAR

Do lokální sitě robota RUDA je zapojen LIDAR **SICK lms111-10100** (viz sekce 2.2.1) přes ethernetový kabel. Pro získání dat ze samotného LIDARu do sytému ROSu jsem použila ovladač (přímo balík v ROSu) pro serii LIDARů SICK LMS1xx. Jedná se o balík **`LMS1xx`**, který je dostupný od verze ROSu Groovy až po nejnovější Kinetic. Pro správný chod tohoto ovladače je potřeba nastavit minimálně parametry zmíněné v tabulce 4.1. Výstup (laserové

| parametr | typ | výchozí hodnota |
|---|---|---|
| host | string | 192.168.1.2 |
| frame_id | string | laser |

Tabulka 4.1: Nastavitelné parametry pro ovladač LIDARu v ROSu.

skeny) potom můžeme číst ve formátu `sensor_msgs/LaserScan` na kanálé `/scan`.

### 4.4.2 Stereo kamera

Se stereo kamerou je nastavení a zprovoznění poněkud složitější. Vzhledem k tomu, že používáme kamery od výrobce Prosilice (viz sekce 2.2.1), měl by robot mít hardwarové rozhraní přizpůsobeno rychlostem kamer. To znamená, že obě kamery by měly mít k dispozici gigabitový ethernetový kabel (GigE). Jedna z nich ale toto kritérium nesplňuje, což nám přináší komplikace především na straně **ROS ovladačů** pro tento typ kamery, kde autor ovladače automaticky předpokládá s gigabitovou rychlostí a hrubé přenastavení parametrů není postačující. Proto je potřeba udělat několik opatření. Použili jsme tedy konkrétně balík **`prosilica_camera`**.

1. Před jakýmkoli nastavením ROS balíku (ovladače) jsem nejprve provedla interní nastavení kamery. A to za pomoci software přímo od výrobce, Vimba [6]. Zde je potřeba si dát především pozor na správné nastavení parametru **`StreamBytesPerSecond`** - a to právě kvůli tomu, že jedna kamera nesplňuje hardwarový požadavek na rychlost přenosu. Dále je vhodné zde zkontrolovat nastavení samotného přenášeho obrazu a zjistit si IP adresy kamer, které jim byly přiděleny.

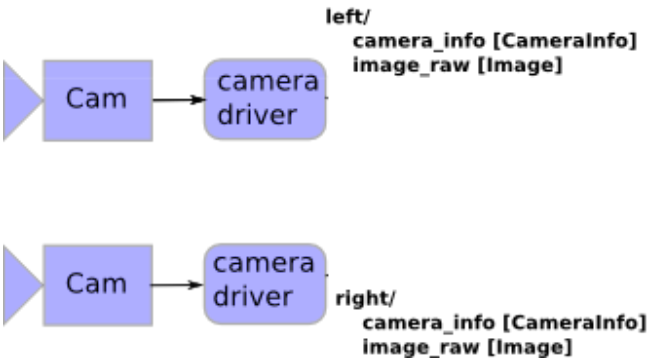
2. Nyní můžeme přejít k ovladači `prosilica_camera` z ROSu. Ten nám umožňuje dostat přímá data z kamery ve vhodném formátu právě do našeho robotického systému, kde s nimi můžeme dále pracovat. Zde byla potřeba zasáhnout přímo do kódu ovladače, protože autor počítá s gigabitovou rychlostí přenosu. Přesněji se jedná o sou-

| parametr | typ | hodnota |
|---|---|---|
| camera | string | prosilica |
| ip_adress | string | 192.168.0.45 |
| trigger_mode | string | polled |
| frame_id | string | high_def_optical_frame |
| stream_bytes_per_second | string | 12400000 |

Tabulka 4.2: Nastavitelné parametry pro ovladač prosilica kamer v ROSu.

[6] Dostupné a ke stažení na https://www.alliedvision.com/en/products/software.html

bor `prosilica_nodelet.cpp`. Po takovém upravení je opět potřeba správně nastavit několik parametrů, které jsou zmíněny v tabulce 4.2. Uzel tohoto ovladače musíme nastavit a spouštět pro každou kameru zvlášť - to nám napovídá už to, že pro daný ovladač je potřeba nastavit IP adresu a šířku přenosového pásma kamery. Data pak dostáváme na výstupech, které jsou znázorněny na obrázku 4.2. Výstupy z jednotli-




Obrázek 4.2: Ukázka vstupů a výstupů ROS ovladače pro Prosilica kamery.

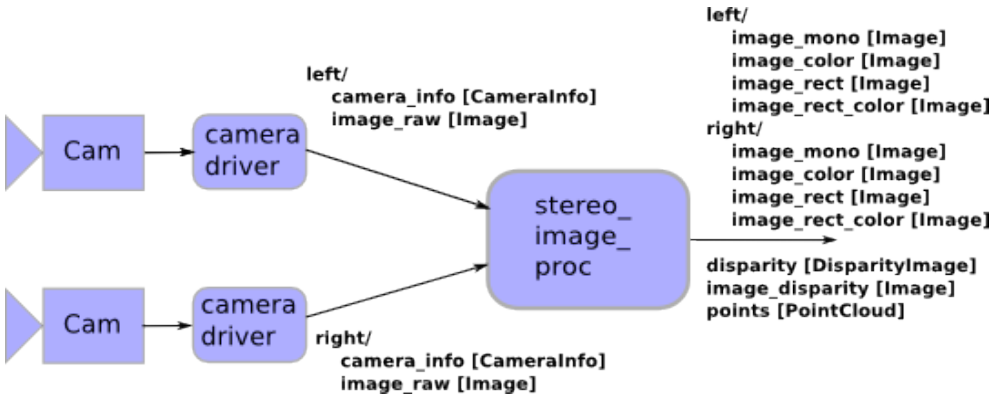
vých kamer tedy čteme na následujících kanálech v tabulce 4.3:

| kamera | typ | namespace | kanál |
|---|---|---|---|
| levá kamera | CameraInfo | /left | /left/camera_info |
| | Image | /left | /left/image_raw |
| pravá kamera | CameraInfo | /right | /right/camera_info |
| | Image | /right | /right/image_raw |

Tabulka 4.3: Tabulka výstupních kanálů (topics) z ovladače kamery.

Dále je pro ovladač potřeba nastavit kalibrační parametry kamery. Ty dáme ovladači na vstup pomocí služby `SetCameraInfo`. **Kalibraci kamer** za účelem získání právě zmíněných kalibračních parametrů kamer jsem prováděla také přímo v ROSu (jinak je vhodné použít například OpenCV, kde pro kalibraci mají také speciální knihovny. Já pro tyto účely použila ROS balík `camera_calibration`. Ten slouží jak ke kalibraci mono kamery, tak ke kalibraci stereo kamery.

Další použité balíky se týkají **stereo zpracování**. Použila jsem multibalík `image_pipeline`, který obsahuje několik vhodných balíků pro práci se stereo daty. Zde je i například zmíněný `camera_calibration`. Ten, který nás bude především zajímat, se jmenuje `stereo_image_proc`. Díky tomuto balíku jsme totiž schopni získat rektifikované snímky (viz sekce 3.2.1). Navazuje přímo na data z `prosilica_camera` a jeho zapojení a komunikaci vidíme na obrázku 4.3. U obrázku jsou přímo vypsány kanály, na kterých je možné výstupní data číst. Například tedy rektifikovaný snímek z pravé kamery vyčteme z kanálu `/right/image_rect` pokud chceme obraz v odstínech šedi a barevný snímek vyčteme z `/right/image_rect_color`. Oba jsou typu `Image`. A právě tato data přijímám na vstup mých vlastních modulů programu - rektifikované a synchronizované snímky z obou kamer v odstínech šedi. Co se týče disparitní mapy, tak tu už z výstupu `stereo_image_proc` nečtu, neboť si tuto fázi zpracovávám sama s pomocí knihovny OpenCV.




Obrázek 4.3: Zapojení balíku `stereo_image_proc` do zpracování dat z kamery.

## 4.5 Modul GMapping

Jak jsme si již řekli, budeme stavět základ lokalizace a mapování na LIDARu. K tomu jsem použila modul `GMapping`, který je volně dostupný pro komunitu SLAMu pod tzv. **OpenSLAM**[7]. Balík je pak v ROSu dostupný pod názvem **`slam_gmapping`**. Metoda SLAM v tomto modulu je založena na částicovém filtru, kde si každá částice vytváří svoji mapu okolí. To je ale velmi výpočetně náročné (částic může být například 100-1000), a proto je zde cílem redukovat tento počet částic. V `GMappingu` právě za tímto účelem používají speciální adaptivní techniku pro učení těchto map z laserových dat. Tento modul k problému přistupuje tak, že přesný výpočet distribuce částic nezáleží pouze na pohybovém modelu robota, ale i na nejnovějším pozorování. Podobná metoda je popsána právě v [38]. Autoři přesné implementace v modulu `GMapping` popisují svůj přístup v [14, 13]. Takový přístup velmi sníží nejistotu pozice robota v predikčním kroku filtru.

I v tomto balíku (`slam_gmapping`) jsem provedla lehkou úpravu kódu - přesně v souboru `slam_gmapping.cpp`. Do publikované zprávy mapy, která se může publikovat při každém skenu, jsem místo časové známky `ros::Time::now()` nastavila právě čas aktuálního příchozího skenu, na základě kterého se mapa počítá. Právě tomuto času by totiž mapa měla být aktuální a zároveň tento údaj budeme dále potřebovat pro synchronizaci, viz sekce 4.7.

Nastavení tohoto modulu je opět nutné provádět pro konkrétní model laserového senzoru - v našem případě tedy SICK lms111-10100. Kromě definování několika soustav, ze kterých si modul potřebuje číst informace (`base_frame, odom_frame`) a soustav, kam bude posílat výstup (`map_frame`), je také důležité definovat parametry samotného algoritmu pro `GMapping`. V tabulce 4.4 jsem zmínila ty nejdůležitější, zbytek je pak nadefinován ve spouštěcím souboru `gmapping_sick.launch` pro senzor na robotu RUDA a `gmapping_hokyo.launch` pro model Hokuyo UTM-30LX, který byl používán v testovacích záznamech.

Uzel dále naslouchá na serveru transformací, tedy na kanále `/tf`, kde potřebuje číst dvě důležité transformace. A to `<frame_příchozích_skenů> -> <báze_robota>` a `<báze_robota> -> <odometrie>`. Bez těchto transformací se samozřejmě výpočet SLAMu neobejde, neboť by nevěděl, jak se robot aktuálně pohybuje a kde na robotu je umístěn LIDAR senzor. Svůj výstup pak publikuje jako transformaci `<mapa> -> <odometrie>`. Dále naslouchá na kanále `/scan`, odkud čte příchozí data ze senzoru. Mapu obsazenosti si pak můžeme vyčíst na kanále `/map`.

[7] Dostupný na adrese http://openslam.org/gmapping.html

| parametr | význam | jednotky | hodnota |
|---|---|---|---|
| maxRange | maximální dosah paprsků senzoru | metry | 20.0 |
| maxUrange | maximální dosah paprsků použitý pro tvorbu mapy | metry | 20.0 |
| sigma | odchylka pro matchování skenů | buňky | 0.05 |
| linearUpdate | robot zpracuje nové měření pouze pokud se pohne minimálně o tolik | metry | 0.0 |
| angularUpdate | robot zpracuje nové měření pouze pokud se natočí alespoň o tolik | radiány | 0.0 |
| particles | počet částic pro filtr | částice | 30 |
| resampleThreshold | práh pro převzorkování částic | relativní | 0.5 |

Tabulka 4.4: Důležité parametry pro správné fungování SLAMu v modulu `GMapping`.

## 4.6 Uzly, komunikace, třídy

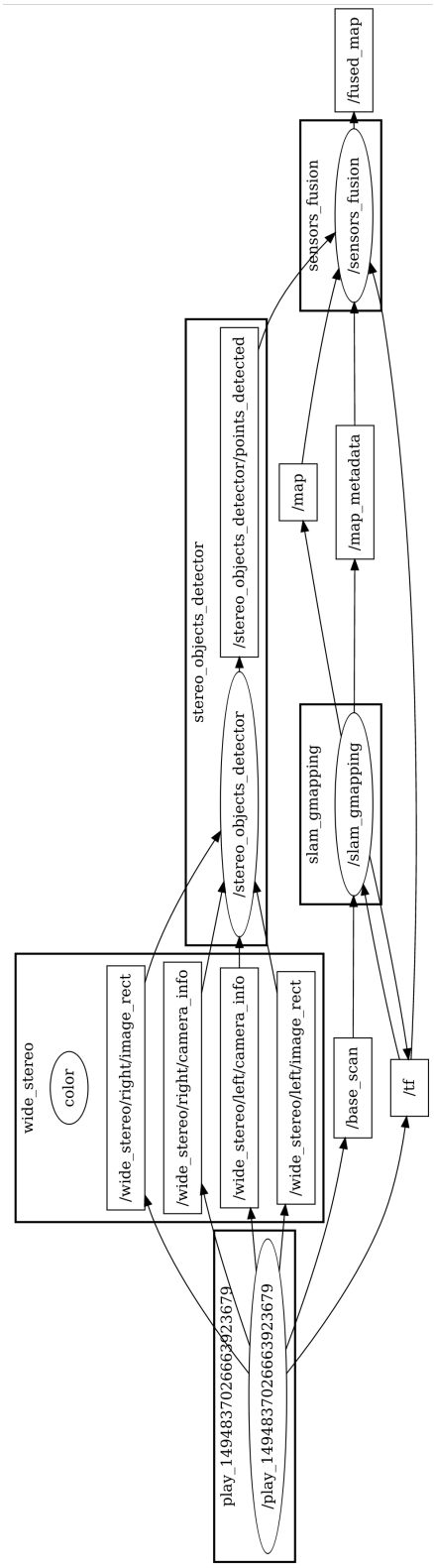
V této sekci si popíšeme, jak vypadá celý implementovaný systém, jak spolu jednotlivé moduly komunikují, kde najdeme výstupy programu a rozebereme podrobnější implementaci přímo z kódu C++.

Celkově se celý systém pro lokalizaci a fúzi dat skládá ze 4 hlavných uzlů (nodes):

1. **`slam_gmapping`** - na vstupu má data z LIDAR senzoru, na výstup dává 2D mapu obsazenosti a pozici robota v mapě,
2. **`stereo_objects_detector`** - na vstupu očekává rektifikované a hardwarově synchronizované snímky ze stereo kamery, na výstup posílá body v 3D ortogonálním systému kamery, které jsou rozlišeny na překážky a volná místa,
3. **`sensors_fusion`** - na vstupu tohoto modulu je mapa obsazenosti z lidar SLAMu a body (už rozlišené na překážky a volný prostor) ze stereo kamery, na výstup pak publikuje fúzovanou 2D mapu obsazenosti z dat z LIDARu a stereo kamery,
4. **`play_{x}`** - je záznam (spuštěný `.bag` soubor) nahraných dat ze senzorů včetně odometrie - tyto jednotlivé části publikuje zvlášť na kanály.

| kanál | typ zprávy | význam |
|---|---|---|
| /wide_stereo/left/image_rect | sensor_msgs/Image | rektifikovaný snímek z levé kamery |
| /wide_stereo/right/image_rect | sensor_msgs/Image | rektifikovaný snímek z pravé kamery |
| /wide_stereo/left/camera_info | sensor_msgs/CameraInfo | především kalibrační parametry |
| /wide_stereo/right/camera_info | sensor_msgs/CameraInfo | především kalibrační parametry |
| /base_scan | sensor_msgs/LaserScan | sken z LIDARu |
| /tf | tf/tfMessage | transformace laseru, báze, odometrie |

Tabulka 4.5: Přehled komunikačních kanálů, na kterých naslouchá náš systém modulů.




Obrázek 4.4: Graf všech uzlů v systému včetně jejich komunikace.

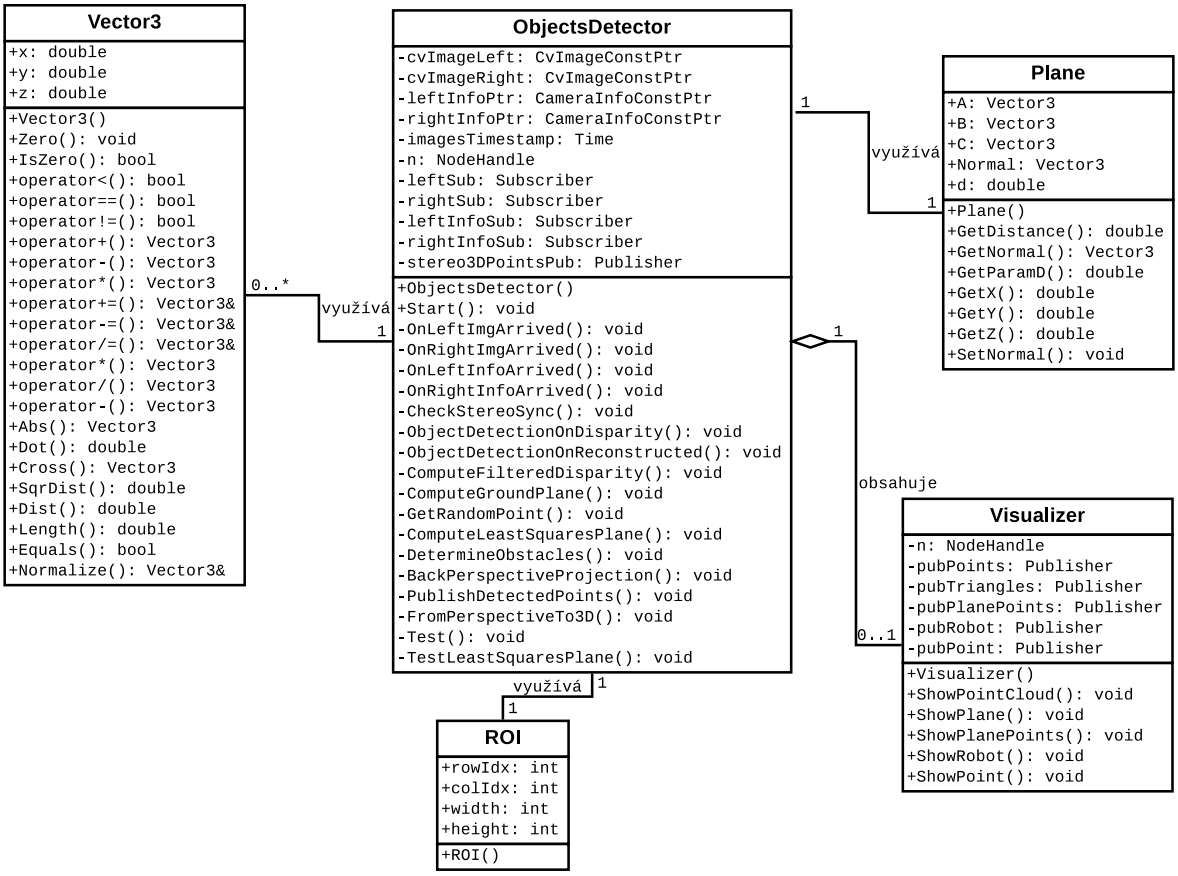
Pro přehled všech uzlů a komunikace mezi nimi zde máme diagram 4.4. V elipsách se nachází názvy uzlů a v menších obdelnících pak kanály, na kterých naslouchají, nebo na kterých publikují. Šipky znázorňují směr komunikace. Větší obdelníky značí jmenné prostory.

Modul `slam_gmapping` jsme si popsali v kapitole 4.5. Dále `play_{x}` je záznam v souboru vytvořený ze senzorů za jízdy robota, na jehož výstupu očekává náš program kanály (přesné definice) z tabulky 4.5. Klíčové uzly si rozebereme podrobněji.

### Modul `stereo_objects_detector`

A zde se konečně dostáváme k těm nejdůležitějším částem implementace. Modul `stereo_objects_detector` pokrývá implementaci práce s obrazem, přesněji od zpracování dat ze stereo kamery po detekci překážek.

Veškerá implementace je zapouzdřena ve třídě `ObjectsDetector`. Obsahuje několik statických konstant pro nastavení jednotlivých algoritmů - přesněji pro bilaterální filtr (`DIAMETER`), výpočet disparity (`SAD_WINDOW_SIZE`), post filtraci disparitní mapy, metodu RANSAC a metodu nejmenších čtverců. Dále jsou zde zakomponovány i některé jednoduché testovací funkce, které mi sloužili pouze pro otestování mnou napsaných komplikovanějších matematických metod. Jedná se o metodu `Test()`, která v sobě vždy volá požadovanou testovací metodu (například `TestLeastSquaresPlane()`). Hlavní část třídy pokrývají metody zpracovávající hlavní proces zpracování dat ze stereo kamery.




Obrázek 4.5: Diagram tříd modulu `stereo_objects_detector`.

Celý proces stereo zpracování se spustí, pokud funkce `CheckStereoSync()` neselže. Ta kontroluje, zda poslední příchozí snímky z levé a pravé kamery mají stejnou časovou známku. Pak spouští hlavní funci
`ObjectDetectionOnDisparity()`, která v sobě skrývá stěžejní část procesu. Volá sekvenčně následující metody:

1. `ComputeFilteredDisparity()` - má za cíl spočítat filtrovanou mapu disparity. Na svém vstupu má již (i softwarově ověřené) synchronizované snímky ze stereo kamery. Snímky nejprve vyhladím pomocí bilaterálního filtru (`bilateralFilter()` z OpenCV), abychom se částečně zbavili šumu. Filtr je sice dvouprůchodový (takže má náročnější zpracování), ale za to přesnější než ostatní filtry. Navíc nám v této implementaci o rychlost zpracování nešlo. Dále pro získání disparitní mapy používá algoritmus matchování bloků z OpenCV - `StereoBM::StereoBM()`, který v sobě zapouzdřuje funkci pro nalezení korespondencí mezi snímky. Po získání disparitní mapy ještě provádím post filtrování, díky kterému obnovím hrany v disparitním obraze z původního vzorového. K tomu využívám třídu `DisparityWLSFilter` opět z OpenCV. Hotová mapa je pak na výstupu mojí funkce pro získání filtrované disparity.

2. `ComputeGroundPlane()` - tato funkce má za úkol v disparitní mapě najít pozemní rovinu, po které se robot poybuje. Nejprvě používám vlastní implementaci metody RANSAC, kterou jsem popsala v sekci 3.2.3. Pro tento účel používám geometrický model roviny, který hledám dynamicky v každém okamžiku. Díky vhodné ukončující podmínce se mi daří nalézt hrubý odhad roviny velmi rychle. Zároveň je důležité poznamenat, že nehledám rovinu v celém obrázku, ale pouze v oblasti zájmu. Touto oblastí jsem zvolila obdelník o rozměrech dolní poloviny snímku a na šířku pak 3/5 snímku. Na metodu RANSAC navazuje metoda nejmenších čtverců, jejíž detail je vysvětlen v sekci 3.2.3. Implementaci najdeme ve funkci `ComputeLeastSquaresPlane()`. Zde nejprve aplikuji minimalizační metodu na součet čtvercových vzdáleností bodů od roviny, poté pomocí řešení problému vlastních vektorů a vlastních hodnot najdu optimální řešení. Pro práci s vlastními hodnotami a vlastními vektory používám knihovnu **Eigen**. Výstupem funkce pro výpočet roviny je námi definovaná struktura `Plane`, která obsahuje hodnoty všech parametrů roviny potřebných pro její přesnou definici.

3. `DetermineObstacles()` - metoda určuje, které body v disparitní mapě jsou překážkou, a které jsou volně průjezdnou oblastí pro robota. Detailní popis metody najdete v sekci 3.2.4. Aneb zvolím si dva prahy a testuju, do které třídy (rozdělené podle prahů) bod zařadím. Na vstupu metody je disparitní mapa a rovina nalezená metodou RANSAC, na výstupu pak dvě množiny bodů, z nichž jedna obsahuje překážky a druhá body definující volný prostor pro robota.

4. `FromPerspectiveTo3D()` - na vstupu metody jsou všechny body z předchozí metody, které jsou vlastně zkresleny perspektivní projekcí, viz sekce 3.2.5. Na základě získaných parametrů ze stereo kamery spočítá metoda `BackPerspectiveProjection()`, která používá třídu `image_geometry::StereoCameraModel` z ROSu, zpětnou perspektivní projekci. Na výstup tedy dává body převedené z perspektivní projekce (disparitní mapy) do 3D ortogonálního systému souřadnic stereo kamery.

5. `PublishDetectedPoints()` - je metoda, která pouze publikuje výsledné body na kanál `/stereo_objects_detector/points_detected`. Jedná se o vlastní vytvořenou

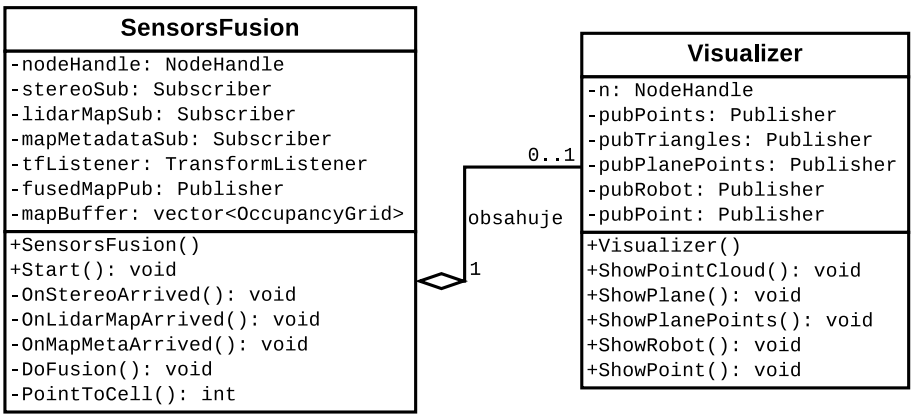
zprávu `DetectedPoints.msg`, ve které najdeme pole bodů reprezentujících překážky (`geometry_msgs/Point[] obstaclePoints`) a pole bodů reprezentujících volné oblasti (`geometry_msgs/Point[] freePoints`).

### Modul `sensors_fusion`

Tento modul se stará o fúzi dat. Přesněji má na vstupu data ze SLAMu pomocí laserového senzoru LIDAR a informace (množiny bodů) z výše popsaného modulu `stereo_objects_detector` představující překážku, nebo volnou oblast. Na výstup má modul za úkol posílat jednu fúzovanou 2D mapu z dat obou senzorů.

Sestává z jedné hlavní třídy, a to **`SensorsFusion`**. Funkce (callback), která startuje fúzi, se jmenuje `OnStereoArrived()`. Proces fúze začíná na základě příchozích informací od stereo kamery proto, že má kamera menší frekvenci než LIDAR, proto se LIDAR může v tomto případě podřídit. Metoda zkontroluje, zda je vše v pořádku, potom volá metodu `DoFusion()`, která sestává z těchto hlavních kroků:

1. nejprve nalezneme na tzv. TF serveru (server, kam se publikují transformace), kde se robot (respektive kamera) aktuálně nachází v mapě (ze SLAMu laseru),
2. načteme tuto mapu do struktury, abychom do ní mohli přidat právě data z kamery,
3. na základě zjištěné transformace mezi kamerou a mapou jsme schopni přidat naměřená data z kamery (pouze body definující pozice překážek) na přesné pozice do mapy, viz postup v sekci 3.2.7,
4. na výstup posíláme mapu, která má v sobě fúzi dat z LIDARu a stereo kamery, na kanál **`/fused_map`**.




Obrázek 4.6: Diagram tříd modulu `sensors_fusion`.

### Balík `my_utils`

Tento balík je vytvořen za účelem používání mých různých pomocných tříd, včetně vizualizace. Každá třída je zvlášť v souboru. Popišme si stručně, které třídy balík obsahuje a k čemu se používají:

1. `Visualizer` - třída, která implementuje několik metod pro vizualizaci různých mezivýsledků v prostředí RViz. V následující tabulce 4.6 vidíme, co která metoda vizualizuje a na jaký kanál:

| metoda | kanál | co vizualizuje |
|---|---|---|
| ShowPointCloud() | /visualization/points | body ve 3D prostoru |
| ShowPlane() | /visualization/plane | nalezená pozemní rovina |
| ShowPlanePoints() | /visualization/plane_points | 3 body definující rovinu |
| ShowRobot() | /visualization/robots_pose | pozice robota v mapě |
| ShowPoint() | /visualization/point | libovolný bod v mapě |

Tabulka 4.6: Přehled vizualizačních metod, jejichž výstupy lze shlédnout v RVizu.

2. `Plane` - třída, která obsahuje možnost definice geometrického modelu roviny různými způsoby tak, aby byla jednoduše použitelná v celém kódu. Navíc obsahuje některé metody pro práci s rovinou, jako například `GetDistance()` - funkce počítající vzdálenost bodu od roviny.

3. `Vector3` - jednoduchá třída, která má za úkol ulehčit práci s body v trojrozměrném prostoru, či trojprvkové vektory. Implementuje všechny základní operace včetně rozšíření, jako například vektorový součin, velikost vektorů, normalizovaný vektor apod...

4. `ROI` - primitivní třída která pouze dopočítá na základě udajů o indexech obdelníku v obraze rozměry, které budou použity jako oblast zájmu.

## 4.7 Synchronizace dat ze senzorů

Vzhledem k tomu, že LIDAR senzor a stereo kamera snímají na **rozlišných frekvencích**, je potřeba čtení a práci s daty nějakým způsobem synchronizovat. Frekvence LIDARu je **40Hz**, zatímco frekvence kamery je **10Hz**. Při fúzi tedy musíme vždy najít vhodná data, která spolu zpracovat. Díky TF serveru v ROSu (server, kam se posílají veškeré transformace) můžeme vyhledávat transformace i do určité hloubky historie. Co se tedy transformací týče, ty zpětně najdeme. Co ale potřebujeme zajistit je, že když přijdou nová data z kamery, budeme schopni přistoupit k datům z LIDARu s nejbližší časovou známkou razítku z kamery.

Pro tuto situaci jsem implementovala synchronizační buffer, kam se mi průběžně ukládají mapy z LIDARu (časově uspořádané za sebe):

```
vector<nav_msgs::OccupancyGridPtr> mapBuffer.
```

Nad tímto bufferem pak provádím primitivní algoritmus - každou příchozí mapu z LIDAR SLAMu přídám do bufferu a při příchozích informacích ze stereo kamery použiju mapu (ukazatel) z bufferu, jejíž časové razítko je starší než u mapy LIDARu, ale zároveň z těch starších záznamů je to ten nejnovější. Při nalezení vhodné mapy všechny starší mapy z bufferu odstraním (aktuální použitou ponechám kvůli bezpečnosti), čímž zajistím, že buffer nepřeteče a zároveň bude pro každou další informaci ze stereo kamery dostupná v bufferu alespoň jedna mapa z LIDAR SLAMu.

# Kapitola 5

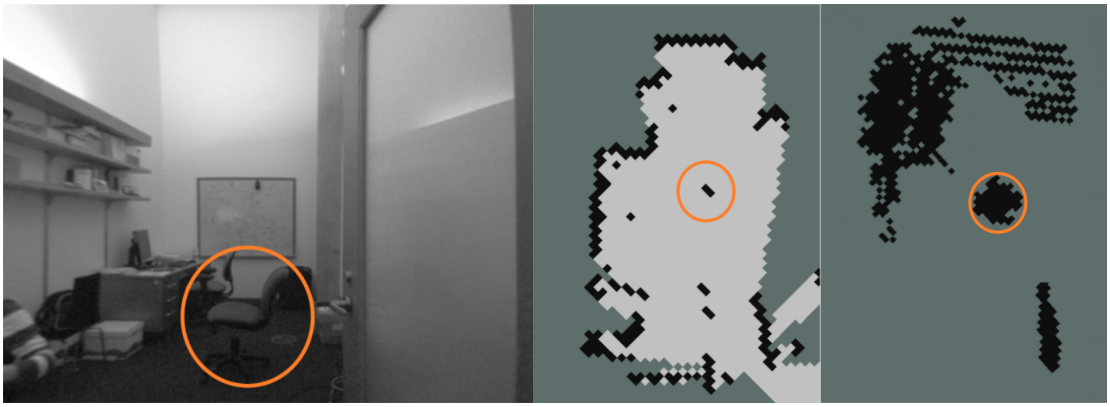
# Testování a vyhodnocení

Pro testování naší metody jsem použila několik scénářů. Jednak jsem testovala pouze některé části implementace, abych ověřila, zda je vhodná pro zpracování dat robota v reálném čase. Největší pozornost jsem však věnovala části, jak moc funguje naše připojení dat ze stereo kamery do LIDAR mapy, zda jsou získané informace užitečné, zda dávají smysl a jak moc naopak dokážou uškodit.

## 5.1 Fúze dat z LIDARu a kamery

Na vstupu tedy máme 2D mapu obsazenosti z LIDARu a data ze stereokamery. Nejprve jsem testovala, jak tedy vypadá mapa z LIDAru, naproti tomu jsem porovnala mapu z kamery. Pro přehlednost zde tyto dvě mapy zobrazuji zvlášť, aby bylo zřejmé, co který senzor za informaci přináší. Spojení těchto map pak probíhá jednoduchým způsobem popsaným v sekci 3.2.7. Představím zde několik různých scén, na kterých si ukážeme různé případy. Na každé scéně budou 3 obrázky vedle sebe - vlevo bude zdrojový snímek, uprostřed mapa vytvořená LIDARem a vpravo bude mapa vytvorená z dat stereo kamery.

Pro část experimentů byla použita datová sada *MIT Stata Center Data Set*, kde používají LIDAR Hokuyo UTM-30LX a Willow Garage Stereo kameru. Záznamy jsou dostupne online[1] a podrobnější informace naleznete například v [8].



Obrázek 5.1: Scéna s židlí - zdrojový obrázek vlevo, mapa z LIDARu uprostřed, mapa z kamery vpravo.

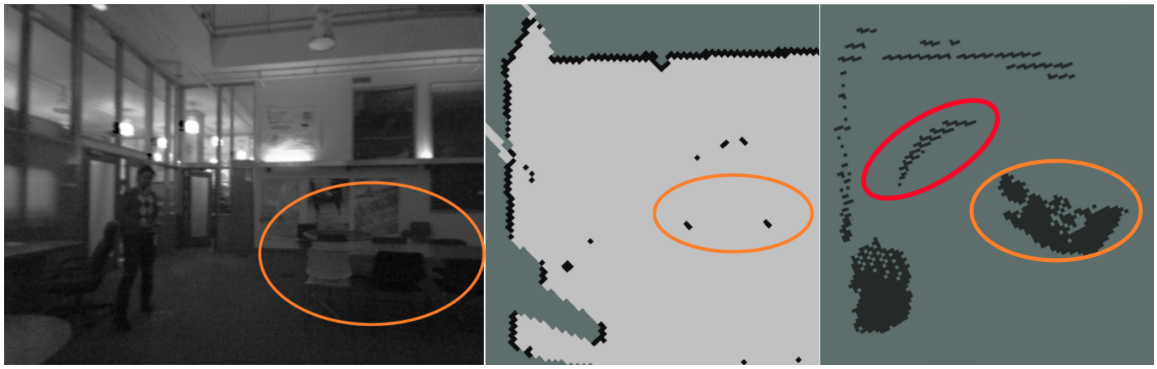
[1] Na adrese http://projects.csail.mit.edu/stata/downloads.php

Na obrázku 5.1 vidíme situaci, kde se robot nachází ve dveřích kanceláře. Na mapě z LIDARu si můžeme všimnout, ze je tento senzor hodně přesný. Nicméně, jak jsem zmínila v textu už dříve, LIDAR vidí jen v jedné rovině, a proto z celé židle uprostřed místnosti vidí pouze její nohu (znázorněno oranžovou elipsou). Robot je ale vyšší než židle, proto pro robota představuje překážku celá židle a nejen ta noha. Tuhle situaci nám řeší mapa ze stereo kamery (vpravo). Tam sice vidíme, že kamera je zase oproti LIDARu relativně nepřesná, data má mírně posunutá, ale detekuje překážky, které LIDAR nevidí. Židle (tentokrát v celé šířce) ze opět znázorněna oranžovou elipsou.

Obrázek 5.2: Scéna na chodbě - zdrojový obrázek vlevo, mapa z LIDARu uprostřed, mapa z kamery vpravo.

Další vyobrazenou situací je robot v dlouhé úzké chodbě (obrázek 5.2). Tady je kamera ukázkou toho, že ne vždy musí být tato metoda přínosem. LIDAR nemá s detekcí okolí problém, dokonce vidí až na konec chodby (25-30 metrů) a to všechno stále velmi přesně. Oproti tomu kamera (jak je vidět v oranžové elipse) má **se vzdáleností problém**. V takovýchto délkách už je velmmi nepřesná, a proto je potřeba si na tyto situace dát pozor (třeba za pomoci filtru pro určité vzdálenosti). Co si na této scéně dále můžeme všimnout, je i nepřesná detekce (v LIDARu přímých) stěn nalevo a napravo od robota. To je způsobeno nepřesnou disparitní mapou (stejná situace ale disparitní mapa viz 3.8) - zde je vidět, že pozemní rovina je mírně prohnutá, a proto pro detekci roviny musel být povolen práh vzdálenosti bodů od roviny. Tím nám ale do definice roviny pak spadají i některé body nad rovinou. Další problém je homogenita obou stěn, které pak rozložení bodů na zdi zašumí a rozhází. Ve výsledku nám metoda vrátí právě to, co je na obrázku.

Na obrázku 5.3 vidíme situaci podobnout scéně se židlí. Ve zdrojovém snímku (s mírně sníženou viditelností) je v oranžové elipse stůl a okolo něj jsou židle, které maji tenké lesklé nohy. V mapě z LIDARu vidíme, že jediné, co detekoval laser, jsou 2 tlusté nohy od stolu, ale nezaregistroval ani jednu židli. To je z toho důvodu, že židle jsou lesklé a odráží paprsky do stran, takže se k LIDARu dostanou špatná data (nebo žádná). Dále má stůl nohy tak daleko od sebe, že by mezi nimi robot bez problému projel. Opět zde ale zavazí robotovi deska stolu, která už je překážkou. Zde je nám opět nápomocí kamera (obázek vpravo), kde sice nepřesně, ale detekovala stůl a i některé židle. Speciálním případem v tomto snímku je tzv. **false positive** (chybně nalezen) objekt znázorněný v červené elipse v mapě kamery. Důvod jeho nálezu je opět problém s disparitní mapou, která je v tomto snímku mírně prohnutá a má nepřesný sklon. Proto se pak část pozemní roviny detekuje jako překážka.



Obrázek 5.3: Scéna se stolem - zdrojový obrázek vlevo, mapa z LIDARu uprostřed, mapa z kamery vpravo.

Poslední takto testovaná scéna je vyobrazena na snímcích 5.4. Zde se jedná o případ, kdy může být takováto fúze velmi užitečná pro plánování trasy robota. Paprsky laserového senzoru jsou totiž zastíněny překážkou a nevidí za ni. Na obrázku (vlevo) vidíme nástěnkovou stěnu (na zemi v dolní části oranžové elipsy), za kterou právě náš LIDAR nevidí. Stereo kamera je umístěna na robotu výše, takže jednak trochu za překážky vidí a navíc je schopna rozpoznat svislou stěnu. Z toho pro LIDAR (prostřední obrázek) plyne, že neví, co se za překážkou nachází. Naopak kamera z této pozice už ví, že za nízkou překážkou další překážka a robot tam neprojede. Tudíž je pak schopna ušetřit plánovači trasy počítání a robotu cestu, aby totéž místo musel prozkoumat laserový senzor.

Obrázek 5.4: Scéna s bariérou - zdrojový obrázek vlevo, mapa z LIDARu uprostřed, mapa z kamery vpravo.

Z výše zmíněných příkladů vidíme, že může být stereo kamera v kombinaci s LIDAR senzorem velmi užitečná. Ačkoliv je kamera velmi nepřesná a její data obsahují velké množství šumu, je schopna vidět za překážky, kde má LIDAR zakrýtý obzor a dále vidí i celé tělo objektů, které může v pohybu robota ohrozit. Velkým **problémem** však v našem přístupu zůstává výpočet **disparitní mapy** především na pozemní rovině. Berme ale v úvahu, že podrobná implementace disparitní mapy není cílem této práce. Již existuje mnoho metod, jak získat v takovémto náročném prostředí lepší disparitní mapu a to především v homogenních oblastech. Může to být například předpoklad, že náklon roviny se pod robotem příliš nemění (což ale ve venkovním prostředí nemusí být pravidlem), či různými metodami segmentace obrazu, kde se dá k výpočtu disparitní mapy leccos napovědět.

Dále nám výpočet disparitní mapy také určuje **složitost** této metody. Obecně totiž při výpočtu disparity probíhá proces popsaný v sekci 3.2.2, který nastavuje složitost na hodnotu

$$O(n^2). \tag{5.1}$$

kde $n$ je rozlišení vstupního snímku. I zde ale existují různé optimalizace, takže i v tomto místě se dá rychlost výpočtu zlepšit.

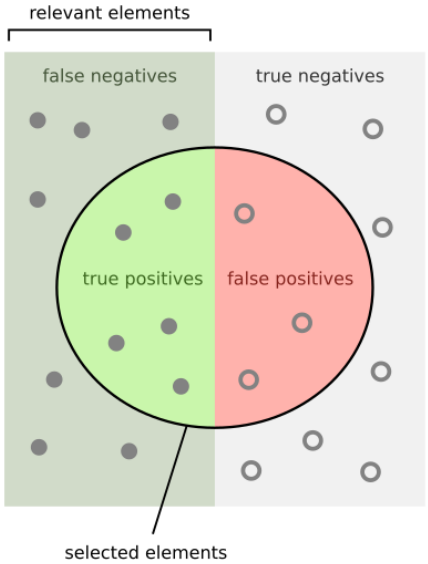
Jednou z výhod našeho přístupu je **rychost získání hrubého odhadu roviny**. Zde jsme použili právě metodu RANSAC (viz sekce 3.2.3). Naše zdrojové obrázky jsou o velikosti 572x460 pixelů. Z toho nás v dolní části obrázku zajíma pro detekci roviny pouze oblast o rozměrech 344x152 pixelů. Tato oblast pro detekci roviny stačí a v následující tabulce 5.1 vidíme, kolik iterací cyklu je pro odhad roviny potřeba:

| | |
|---|---|
| **počet prohledávaných snímků** | 10 000 |
| **rozměr oblasti [px]** | 52288 |
| **průměrný počet iterací [px]** | 21.169 |
| **nejméně iterací** | 8 |
| **nejvíce iterací** | 39 |

Tabulka 5.1: Ukázka síly metody RANSAC pro nalezení roviny v obraze.

### 5.1.1 Přesnost a citlivost

Ještě jsem prováděla jednu metodu vyhodnocení kvality našeho přístupu, a to výpočet tzv. **precision** (přesnost) a **recall** (citlivost). Mějme ale na paměti, že moje implementace je pouze prototyp a neobsahuje žádné optimalizace a je spousta míst, kde by se dala metoda vylepšit. To je důležité si před tímto vyhodnocením uvědomit. Tato metoda vyhodnocení




Obrázek 5.5: Množiny figurující ve vyhodnocení citlivosti a přesnosti[2].

[2]Obrázek převzat z https://en.wikipedia.org/wiki/Precision_and_recall

se nejčastěji používá při rozpoznávání vzorů a klasifikace objektů (kde máme nějakou anotovanou množinu a můžeme strojově rozhodnout, zda byl objekt oklasifikován správně). Její princip je znázorněn na obrázku 5.5.

Pokud si tyto množiny správně definujeme i pro náš případ, můžeme toto vyhodnocení použít. Definujme si tedy množiny následovně:

- **true positives** - oblasti (objekty - např. stůl, židle) ve snímcích, které měly být detekovány jako překážka a jsou detekovány jako překážka,
- **false negatives** - objekty, které měly být detekovány jako překážka ale nebyly detekovány vůbec (tedy je z nich volná oblast),
- **false positives** - objekty, které jsou detekovány jako překážka, ale neměly být (správně jsou to volné oblasti),
- **true negatives** - oblasti, které nejsou překážkou a také nebyly určeny překážkou.

Z těchto množin se pak počítají parametry přesnost a citlivost. Pro přesnost si můžeme pomoci otázkou „Kolik označených objektů za překážku (v obrázku 5.5 v kruhu) bylo správně určeno?“ a zapíšeme jako:

$$Precision = \frac{true\_positives}{true\_positives + false\_positives}. \tag{5.2}$$

Na citlivost se můžeme zeptat „Kolik správně označených objektů za překážku bylo nalezeno ze všech překážek?“ a zapíšeme jako:

$$Recall = \frac{true\_positives}{true\_positives + false\_negatives}. \tag{5.3}$$

Nyní se vrhněme na samotné vyhodnocené našeho přístupu. Jelikož jsem ale veškeré toto vyhodnocení dělala ručně (neměla jsem žádné anotace, žádnou množinu vzorových řešení, ale sama jsem kontrolovala snímek po snímku), berte prosím ohled na počet testovaných dat. **Testovací sada** obsahovala **200 snímků**, přičemž v každém snímku bylo **0-2 překážky** a celkově tyto snímky pokrývaly **20 různých scén**. Robot se pohyboval uvnitř chodeb a kanceláří - tedy ve vnitřním prostředí. Shrnutí výsledků této testovací sady vidíme v tabulce 5.2:

| | |
|---|---|
| překážek celkem (relevant elements) | 240 |
| správně pozitivní | 223 |
| chybně pozitivní | 76 |
| chybně negativní | 17 |

Tabulka 5.2: Naměřené hodnoty detekce překážek v různých scénách.

Z výše uvedené tabulky vyplývá kromě přesnosti a citlivosti několik faktů. Překážky, které má metoda najít, najde skoro vždy (řádek správně pozitivních). Ty, které to vynechalo (chybně negativní) byly v testovací sadě ve dvou scénách, kde bylo velmi zhoršené osvětlení. Disparitní mapa objektů pak splývala s pozadím a proto byla detekce chybná. Naopak číslo, které bychom chtěli výrazně snížit, je hodnota chybně pozitivních nálezů.

Tyto chyby jsou opět způsobeny nedokonalou disparitní mapou a v budoucnu nebude složité je odladit. Jedná se o problém získání disparitní mapy v homogenních regionech, kde může být nápomocí například segmentace obrazu. Jinak, kvůli deformované pozemní rovině, může výsledná nalezená rovina dostat mírně odlišný sklon a pak jsou některé body roviny prohlášeny jako překážka (viz situace na snímku 5.3 vpravo). Přesnost nám tedy vychází

$$\textbf{Precission} = \frac{223}{223 + 76} = 74\% \tag{5.4}$$

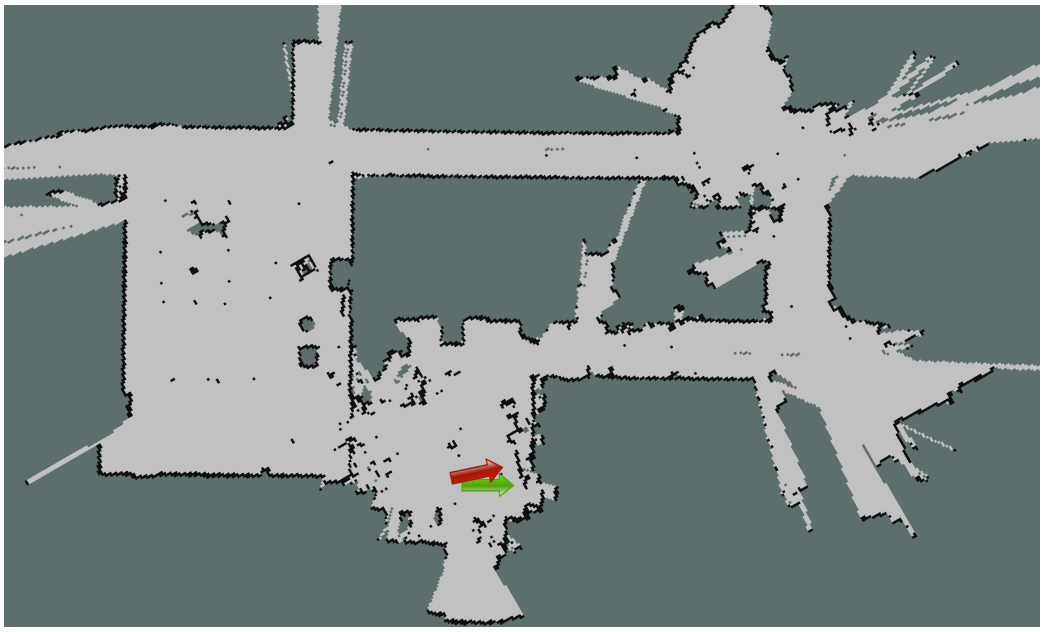
a citlivost

$$\textbf{Recall} = \frac{223}{223 + 17} = 93\%. \tag{5.5}$$

## 5.2 Přínos pro lokalizaci a navigaci

Metodu jsem implementovala za účelem vylepšit lokalizaci a navigaci u mobilních robotů. Mnohé výhody a důvody, proč jsem zvolila právě tuto techniku, jsem zmínila v sekci 3.1. Zde si rozebereme, jaký to má ve výsledku přínos.

Vzhledem k tomu, že je kamera relativně nepřesná, jsme kameru z **lokalizace** vyřadili. Důvody jsou žrejmé už z testovacích snímků v sekci 5.1. Celý náš lokalizační systém tedy funguje pouze na mapování pomocí lidaru. Kamera by zde spíše uškodila. V 90% reálných situací si LIDAR vystačí pro lokalizaci sám. Pouze ve speciálních místech, jako je například planina, poušť, či vyjímečně dlouhá chodba, která by se v mapě jevila pouze jako dvě rovnoběžné úsečky, by mohla kamera v lokalizaci pomoci. Ale i zde už můžou být užitečnější i jiné senzory. Taková testovací data se mi ale nepodařilo sehnat (je za potřebí, aby v záznamech byla stereo kamera, LIDAR a odometrie).



Obrázek 5.6: Zaznamenaný okruh robota pomocí LIDAR SLAMu s výchozí a výslednou pozicí.

Ukažme si tedy alespoň, jak se vypořádá náš aktuální systém se SLAMem v běžném vnitřním prostředí - obrázek 5.6. Robot projel celý okruh napříč několika kancelářemi a chodbami. Zelená šipka znázorňuje výchozí polohu, červená šipka polohu po projetém okruhu (robot se vrátil do výchozí pozice). Ujel zhruba **71 metrů**. Vidíme, že odchylka výchozí pozice od cílové je velmi malá - **43 centimetrů** a **10,2 stupňů**.

Situace ve venkovním prostředí je poněkud složitější (především kvůli dynamickým objektům). Přesnost, které takovýto LIDAR SLAM dosahuje ve venkovním prostředí, je vyčíslena v následující tabulce 5.3. Představuji v ní 3 okruhy, každý po třech měřeních.

| **délka okruhu [m]** | **odchylka vzdálenosti [m]** | **odchylka natočení [rad]** |
|---|---|---|
| 510 | 3 | 0,28 |
| 510 | 5 | 0,33 |
| 510 | 3 | 0,30 |
| 634 | 13 | 0,23 |
| 634 | 18 | 0,30 |
| 634 | 11 | 0,22 |
| 893 | 28 | 0,26 |
| 893 | 33 | 0,24 |
| 893 | 39 | 0,40 |

Tabulka 5.3: Nameřené hodnoty LIDAR SLAMu ve venkovním prostředí.

Testování probíhalo v reálném čase, přesněji o frekvenci 40 skenů za sekundu. Hodnoty v tabulce jsou ale velmi relativní, protože hodně záleží na prostředí, kde byly testy prováděny.

Mnohem zajímavější využití naše metoda nachází v procesu **plánování trasy**. Zde už se vyplatí na vstup plánovače posílat naši fúzovanou mapu především proto, že kamera vidí ohrožující překážky, které LIDAR nevidí - ačkoli mírně nepřesně. Dále dokáže **ušetřit čas** výpočtu i pohyb robota v situacích, kdy si LIDAR ještě myslí, že by mohl někudy projet, ale kamera už ví, že leží v cestě překážka. Plánovač trasy ale nebyl předmětem této práce, a tak zůstává do budoucna, v jakém formátu budeme náš výstup posílat plánovači.

# Kapitola 6

# Závěr

## 6.1 Shrnutí odvedené práce

V první části práce (více teoretické) jsme se seznámili s obecnými přístupy, které se u mobilní robotiky (zejména SLAMu) používají. Poté se v této práci věnuji kritice aktuálního stavu, kde jsme si shrnuli dosavadní používané metody, jejich výhody a nevýhody. Na základě toho zde navrhuji nový přístup pro vylepšení mapování, který může být nápomocí při lokalizaci a plánování trasy robota.

Tento přístup má také své výhody a nevýhody. Cílem práce ale bylo ověřit, zda tedy dokáže být užitečný či nikoliv. Po dokončené implementaci a dostatečném testování jsme ukázali, že tato metoda své uplatnění najde. Ikdyž ne tak moc v lokalizaci, jako v plánování trasy. Zatímco LIDAR není schopen vidět některé překážky, kamera je odhalí a zakreslí do mapy. Ačkoliv je kamera relativně nepřesná, právě pro toto využití (upřesnění, kudy robot může jet) je velmi vhodná. Dokonce může při plánování trasy ušetřit spoustu času v situacích, kdy LIDAR nevidí za překážku a robot by ji musel objíždět, přitom kamera vidí, že ža překážkou je prostor například neprůjezdný.

Výhodami oproti jiným metodám je to, že veškerou detekci překážek provádím z disparitní mapy. Teprve až na konci samotného řetězce provádím rekonstrukci 3D scény, kde už do pozic bodů zanáším nepřesnosti kvůli odhadnutým parametrům získaným při kalibraci kamery. Další výhodou je například přístup k detekci pozemní roviny. Zde totiž využívám dynamický model, který je ve spojení s metodou RANSAC a metodou nejmenších čtverců velmi rychlý a přesný.

Velkým nedostatkem našeho přístupu je výpočet disparitní mapy, kterému jsme nevěnovali tolik času, kolik by bylo potřeba. Ikdyž se daří získat kvalitní výsledky, v některých situacích algoritmus stále chybuje a je jisté, že vylepšení výpočtu disparitní mapy tuto chybovost odstraní, nebo alespoň výrazně zredukuje. Disparitní mapa ale nebyla přímo cílem této práce, proto jsme ji implementovali pouze do fáze vyhovující, nikoliv výborné.

Dále ačkoli to nemusí být na první pohled zřejmé, i zde často narážíme na hranice výkonu. Spousta výpočtů bývá prováděna na grafické kartě, ale vzhledem k tomu, že se bavíme o použití v mobilních robotech, ne vždy si můžeme dovolit tolik prostoru, co je pro výpočty potřeba. Když pak máme několik kamer, které jsou vstupem například do výpočtu lokalizace a navigace a přitom ještě chceme rozpoznávat objekty pomocí nějakého klasifikátoru, musíme už své algoritmy optimalizovat. Ve výpočtech SLAMu je hodně pseudonáhodného generování množství částic, atp... Tyto parametry musíme rozumně omezovat, a to často na úkor kvality algoritmu.

## 6.2 Návrhy do budoucna

Jak jsem již několikrát zmínila, trpí naše implementace na **kvalitě disparitní mapy** (především v homogenních regionech). V budoucnu by se tento nedostatek dal odstranit například za použití segmentace obrazu, viz [18]. Dále by se rozhodně dala lépe propracovat samotná **fúze map** (viz sekce 3.2.7), kde používám naprosto primitivní `OR` operaci. Ve většině případů takovýto přístup opět stačit bude, ale v detailech může pomoci vylepšení. Například bychom mohli dávat LIDARu a kameře nějaké váhy (koeficienty), kterými by se výsledná pravděpodobnost ve fúzované mapě ještě násobila.

Co se týče samotného navázání a pokračování v mém modulu, v budoucnu se předpokládá **implementace plánovače**. Ten bude mít na svém vstupu právě naši fúzouvanou mapu, dle které bude plánovat robotu další trasy. Na straně lokalizace by zase bylo vhodné nasbírat více dat z různých prostředí (vyvézt robota ven do složitějších podmínek) a zkusit aplikovat fúzi právě i na lokalizaci v prostředí, kde je LIDAR naprosto ztracen. Za zkoušku by dále stálo vyhodnotit náročnost a přínosnost našeho zpracování oproti modulu, který by představoval vedle LIDAR SLAMu ještě paralelní kamera SLAM.

# Literatura


[1] Achmad, M. S. H.; Findari, W. S.; Ann, N. Q.; aj.: *Stereo camera — Based 3D object reconstruction utilizing Semi - Global Matching Algorithm*. 2016 2nd International Conference on Science and Technology - Computer (ICST), 2016, ISBN 978-1-5090-4357-6, 194-199 s.

[2] Bay, H.; Tuytelaars, T.; Gool, L. V.: *SURF: Speeded Up Robust Features*. Proceedings of the ninth European Conference on Computer Vision, 2006.
URL http://www.vision.ee.ethz.ch/~surf/eccv06.pdf

[3] Biezen, M.: *Special Topics - The Kalman Filter*. 2015-09-15 [cit. 2016-11-16].
URL https://www.youtube.com/watch?v=CaCcOwJPytQ&list=PLX2gX-ftPVXU3oUFNATxGXY90AULiqnWT

[4] Blow, D.: *To fit a plane to a set of points by least squares*. Acta Cryst, 1960, 168 s., 13. vydání.

[5] Bradski, G.; Kaehler, A.: *Learning OpenCV: Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, 2013, 2. vydání.

[6] Davison, A. J.; Reid, I. D.; Molton, N. D.; aj.: *MonoSLAM: Real-Time Single Camera SLAM*. IEEE Transaction on Pattern Analysis and Machine Intelligence, 2007, 1052-1067 s.

[7] Dubská, M.; Přibyl, B.: *Mapa disparity*. 2012 [cit. 2016-11-23].
URL https://medusa.fit.vutbr.cz/stereo/disparity.php

[8] Fallon, M.; Johannsson, H.; Kaess, M.; aj.: *The MIT Stata Center Dataset*. IJRR Dataset Paper, cit. [2017-04-06], na posouzení.
URL http://projects.csail.mit.edu/stata/index.php

[9] Fernández-Madrigal, J.-A.; Claraco, J. L. B.: *Simultaneous Localization and Mapping for Mobile Robots: Introduction and Methods*. Hershey, PA: Information Science Reference, 2013, ISBN 9781466621060.

[10] Ferrari, F.; Grosso, E.; Sandini, G.; aj.: *A Stereo Vision System for Real Time Obstacle Avoidance in Unknown Environment*. IEEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications, 1990, 703-708 s.

[11] Fischler, M. A.; Bolles, R. C.: *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography*. SRI International, 1981, 381-395 s.

[12] Grewal, M. S.; Andrews, A. P.: *Kalman Filtering: Theory and Practice*. Prentice Hall, 1993, ISBN 013211335X.

[13] Grisetti, G.; Stachniss, C.; Burgard, W.: *Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling*. In Proc. of the IEEE International Conference on Robotics and Automation, 2005, 34-46 s., 23. vydání.

[14] Grisetti, G.; Stachniss, C.; Burgard, W.: *Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters*. IEEE Transactions on Robotics, 2007, 34-46 s., 23. vydání.

[15] Haberdar, H.; Shah, S. K.: *Disparity Map Refinement for Video Based Scene Change Detection Using a Mobile Stereo Camera Platform*. International Conference on Pattern Recognition, 2010, 3890-3893 s.

[16] Hartley, R.; Zisserman, A.: *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003, ISBN 0-521-54051-8.

[17] Kim, J.-E.; Kim, K.-D.; Jung, K.-H.: *Reliable estimation of disparity map in textureless region of roadway*. 2017 19th International Conference on Advanced Communication Technology (ICACT), 2017, ISBN 978-89-968650-9-4, 399-402 s.

[18] Kim, J.-E.; Kim, K.-D.; Jung, K.-H.: *Reliable estimation of disparity map in textureless region of roadway*. 19th International Conference on Advanced Communication Technology (ICACT), 2017, ISBN 978-89-968650-9-4, 399-402 s.

[19] Labayrade, R.; Aubert, D.; Tarel, J.-P.: *Real Time Obstacle Detection in Stereovision on Non Flat Road Geometry Through "V-disparity"Representation*. IEEE Intelligent Vehicle Symposium, 2002, 646-651 s.

[20] Loop, C.; Zhang, Z.: *Computing rectifying homographies for stereo vision*. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1999, ISBN 0-7695-0149-4, 125-131 s.

[21] Lowe, D. G.: *Distinctive Image Features from Scale-Invariant Keypoints*. International Journal ournal of Computer Vision, 2004.
URL http://www.robots.ox.ac.uk/~vgg/research/affine/det_eval_files/lowe_ijcv2004.pdf

[22] Matthies, L.; Grandjean, P.: *Stochastic Performance Modeling and Evaluation of Obstacle Detectability with Imaging Range Sensors*. IEEE Transactions on Robotics and Automation, 1994, 783-792 s., ISSN: 1042-296X.

[23] Min, D.; Choi, S.; Lu, J.; aj.: *Fast Global Image Smoothing Based on Weighted Least Squares*. IEEE Transactions on Image Processing, 2014, 5638-5653 s., ISSN: 1941-0042.

[24] Olson, E.: *A Primer on Odometry and Motor Control*. 2004.
URL https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-186-mobile-autonomous-systems-laboratory-january-iap-2005/study-materials/odomtutorial.pdf

[25] OpenCV tým: *Canny Edge Detector*. 2017 [cit. 2017-01-03].
URL http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html

[26] Orság, F.; Drahanský, M.: *Robot pro hledání osob v závalech a lavinách*. Vysoké učení technické Brně, Fakulta informačních technologií, 2015.

[27] Pearson, K.: *On lines and planes of closest fit to systems of points in space*. Philosophical Magazine, 1901, 559-572 s., 2. vydání.

[28] Radtke, T.; Zerbe, V.: *Tracking of Dynamic Objects Based on Optical Flow*. 2001 [cit. 2016-11-14].
URL https://www.tu-ilmenau.de/fileadmin/public/sse/Veroeffentlichungen/2001/ICIMADE2001%20Paper.pdf

[29] Riisgaard, S.; Blas, M. R.: *SLAM for Dummies: A Tutorial Approach to Simultaneous Localization and Mapping*. Massachusetts Institute of Technology, 2005.

[30] Sasaki, H.; Kubota, N.; Taniguchi, K.: *Growing Topological Map for SLAM of Mobile Robots*. SICE Annual Conference in Japan, 2008.

[31] Schomaker, V.; Waser, J.; Marsh, R.; aj.: *To Fit a Plane or a Line to a Set of Points by Least Squares*. Acta Cryst, 1959, 600-604 s., 12. vydání.

[32] Scratchapixel: *The Perspective and Orthographic Projection Matrix*. 2014 [cit. 2017-05-07].
URL
https://www.scratchapixel.com/lessons/3d-basic-rendering/perspective-and-orthographic-projection-matrix/projection-matrix-introduction

[33] Společnost Allied Vision: *Prosilica GT*. 2016 [cit. 2017-01-05].
URL https://www.alliedvision.com/en/products/cameras/detail/Prosilica%20GT/1290.html

[34] Společnost SICK: *2D laser scanners*. 2017 [cit. 2017-01-05].
URL https://www.sick.com/de/en/detection-and-ranging-solutions/2d-laser-scanners/lms1xx/lms111-10100/p/p109842

[35] Thrun, S.: *Particle Filters in Robotics*. In Proceedings of Uncertainty in AI (UAI), 2002.
URL http://robots.stanford.edu/papers/thrun.pf-in-robotics-uai02.pdf

[36] Thrun, S.; Burgard, W.; Fox, D.: *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[37] Vojáček, A.: *Princip optických enkodérů polohy pro řízení motorů*. 2006 [cit. 2017-01-08].
URL http://automatizace.hw.cz/clanek/2006022801

[38] Vu, T.-D.; Aycard, O.; Appenrodt, N.: *Online Localization and Mapping with Moving Object Tracking in Dynamic Outdoor Environments*. 2007 IEEE Intelligent Vehicles Symposium, 2007, 190-195 s.

[39] Winkler, Z.: *Lokalizace*. 2003-11-03 [cit. 2016-11-12].
URL https://robotika.cz/guide/localization/cs

[40] Winkler, Z.: *Odometrie*. 2005 [cit. 2016-11-12].
URL http://robotika.cz/guide/odometry/cs

[41] Wurm, K. M.; Hornung, A.; Bennewitz, M.; aj.: *OctoMap: A Probabilistic, Flexible, and Compact 3D Map Representation for Robotic Systems*. In Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation, 2010.

[42] Yan, L.; Zhao, X.; ; aj.: *Research on 3D measuring based binocular vision*. 2014 IEEE International Conference on Control Science and Systems, 2014, ISBN 978-1-4799-6397-3, 18-22 s.

[43] Yang, A.; Li, X.; Jia, S.; aj.: *Monocular Three Dimensional Dense Surface Reconstruction by Optical Flow Feedback*. 2015 IEEE International Conference on Information and Automation, 2015, 504-509 s.

# Příloha A

# Obsah CD

| **Umístění** | **Obsah složky** |
|---|---|
| `/README` | Popis obsahu CD a návod na překlad a spuštění |
| `/doc/LIDAR_a_stereo_kamera.pdf` | Text práce ve formátu PDF |
| `/doc/src` | Zdrojové soubory k prácí ve formátu LaTeX |
| `/catkin_workspace/src` | Zdrojové soubory programu |
| `/catkin_workspace/launchfiles` | Spouštěcí soubory programu |

Tabulka A.1: Obsah přiloženého CD.

# Příloha B

# Návod na překlad a spuštění

## Překlad

1. Vytvoříme klasický ROS (catkin) pracovní adresář (`your_catkin_workspace`), viz `http://wiki.ros.org/catkin/Tutorials/create_a_workspace`
2. Vložíme soubory z `/catkin_workspace` v takové hierarchii jak jsou do našeho vlastního catkin workspace (obsah adresáře `/catkin_workspace/src/` tedy bude v něčem jako `your_catkin_workspace/src/`, obdobně s `/catkin_workspace/launchfiles`
3. Celý projekt přeložíme příkazem

   ```
   catkin_make
   ```

   Je možné, že bude potřeba před překladem celého projektu ručně přeložit jeden balíček, a to příkazem

   ```
   catkin_make --pkg=driver_base
   ```

   Je to kvůli špatné hierarchii překladu v jednom z balíků ROSu (může být rozdíl mezi verzemi), který je oficiální, a proto jsem do něj nezasahovala.
4. Adresáře `/launchfiles` a `/src` z CD budou tedy na stejné úrovni

## Spuštění

1. Spustíme jádro ROSu:

   ```
   roscore
   ```
2. Pak spustíme `.bag` soubor s nahranými potřebnými daty, viz kapitola Testování a vyhodnocení (nebo můžeme číst přímo data z robota). Ten musíme pustit s následujícími parametry:

   ```
   rosbag play --clock my_bagfile
   ```
3. Dále pustíme modul `GMapping` pro tvorbu LIDAR mapy. Pro data přímo z robota RUDA spustíme

```
roslaunch launchfiles/gmapping_sick.launch
```

a pro data ze záznamu `.bag` spustíme

```
roslaunch launchfiles/gmapping_hokyo.launch
```

4. Modul pro zpracování dat ze stereo kamery spustíme příkazem

```
roslaunch launchfiles/stereo_objects_detector.launch
```

5. Pro výslednou fúzi spustíme modul `sensors_fusion`:

```
roslaunch launchfiles/sensors_fusion.launch
```

6. Hlavní výstup pak čteme na kanále `/fused_map`, viz kapitola Implementace

Poznámka: Jeden z použitých `.bag` souborů při vývoji (na který jsou i nastaveny určité parametry) stáhnete zde: http://infinity.csail.mit.edu/data/2011/2011-01-18-06-37-58.bag