VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

SIMULTALNÍ LOKALIZACE, MAPOVÁNÍ A VYTVÁŘENÍ MODELU PROSTŘEDÍ PRO AUTONOMNÍ ROBOTIKU

SLAM METHODS AND TOOLS FOR BUILDING WORLD MODELS IN AUTONOMOUS ROBOTICS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Tomasz Pilch

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. Radomil Matoušek, Ph.D.

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Tomasz Pilch**
Studijní program:	Strojní inženýrství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**doc. Ing. Radomil Matoušek, Ph.D.**
Akademický rok:	2015/16

Ředitel ústavu Vám v souladu se zákonem č. 111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Simultalní lokalizace, mapování a vytváření modelu prostředí pro autonomní robotiku

Stručná charakteristika problematiky úkolu:

Metody SLAM (simultální lokalizace a mapování) jsou základní techniky pro zpracování dat a lokalozaci v mobilní robotice, řešící problém konstrukce nebo update map v neznámem prostředí. Předmětem práce bude zpracování dat ze senzoru LIDAR a implementace zvolených algoritmů v kontextu SLAM a tvorby modelu prostředí.

Cíle diplomové práce:

1) Reserse problematiky SLAM a strucny popis zvolených metod.
2) Stručná rešerše senzorů LIDAR a popis zvoleného řešení.
3) Programová implementace (zpracování dat, tvorba mapy atd.).
4) Demonstrace řešení v modelových prostředích, vyhodnoceni.

Seznam literatury:

J. Mullane, B.-N. Vo, M. D. Adams, and B.-T. Vo, (2011): A random-finite-set approach to Bayesian SLAM. IEEE Transactions on Robotics 27 (2): 268–282. doi:10.1109/TRO.2010.2101370

Leonard, J.J.; Durrant-whyte, H.F. (1991): Simultaneous map building and localization for an autonomous mobile robot. Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on: 1442–1447. doi:10.1109/IROS.1991.174711. Retrieved 2008-04-08

Karlsson, N.; Di Bernardo, E.;Ostrowski, J;Goncalves, L.;Pirjanian, P.;Munich, M. (2005): The vSLAM Algorithm for Robust Localization and Mapping. Int. Conf. on Robotics and Automation (ICRA)

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2015/16.

V Brně, dne 7. 12. 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ
DĚKANÁT
L. S.

Ing. Jan Roupec, Ph.D.
ředitel ústavu

doc. Ing. Jaroslav Katolický, Ph.D.
děkan

Abstrakt
Tato diplomová práce pojednává o problematice SLAM (simultalní lokalizace a mapování) a představuje možnosti její implementace. Hlavním cílem je implementovat zvolený algoritmus k tvorbě mapy a zpracovat naměřená data ze senzoru LIDAR.

Summary
This master thesis explains what is SLAM and reprezents possibilities of implementation. The main goal is to implement chosen algorithm to create map and process measured data from LIDAR sensor.

Klíčová slova
Simultánní lokalizace a mapování, SLAM, implementace SLAM, extrahování důležitých charakteristik, LIDAR

Keywords
Simultaneous localization and mapping, SLAM, implementation of SLAM, landmark extraction, LIDAR

PILCH, T.*Simultalní lokalizace, mapování a vytváření modelu prostředí pro autonomní robotiku.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2016. 66 s. Vedoucí doc. Ing. Radomil Matoušek, Ph.D.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně dle pokynů vedoucího a s použitím uvedené odborné literatury.

Bc. Tomasz Pilch

Rád bych poděkoval vedoucímu své diplomové práce doc. Ing. Radomilu Matoušovi, Ph.D. za jeho rady a pomoc při zpracovávání této práce. Také bych rád poděkoval Bc. Ladislavu Dobrovskému za pomoc při implementaci serverové části a UDP komunikace a nakonec bych rád poděkoval své rodině a kamarádům, kteří mi byli oporou jak při psaní této práce, tak i při celém studiu.

Bc. Tomasz Pilch

Obsah

Úvod

Jedním z nejnovějších technických trendů současnosti je vývoj autonomní robotiky, což zahrnuje jak roboty, tak vozidla a další autonomní mechanismy. Autonomní zařízení musí být schopno rozpoznávat a identifikovat objekty a této informace využít ke své orientaci. Ať je pro potřebu lokalizace využito optické kamery, LIDARu nebo sonaru, je ke zpracování dat majoritně využíván algoritmus označovaný jako SLAM. Vstupní data nejsou zcela směrodatná (nezáleží na reprezentaci), protože se vždy provádí stejné kroky. SLAM znamená simultánní lokalizace a mapování a ve většině případů pracuje s důležitými objekty jako jsou rohy a hrany. Na začátku se z naměřených dat (mračna bodů, obrázky, atd.) extrahují významné charakteristiky. Následuje popis jednotlivých objektů (pozice, věrohodnost, typ, atd.), pokud je popis dokončen, může se začít s registrací mračen bodů. Registrací se rozumí aplikace algoritmu, který na základě nalezených objektů nebo měření samotném najde požadovanou transformaci mezi měřeními. Většinou se hledá rotace a translace, aby bylo možno odvodit posuny robota. Opakováním tohoto kroku se skládá mapa okolí. Používá se algoritmů založených na statistice, protože řešit tuto úlohu hrubou silou, by nebylo reálné. Statistika pomáhá najít jakýsi odhad transformací a minimalizuje se chyba. Chybou v tomto případě se myslí podobnost dat. Tyto algoritmy konvergují rychleji, ale je potřeba dávat si pozor na lokální extrémy a chybně změřené body.

Cílem této diplomové práce je seznámit se s algoritmem SLAM a správně zpracovat naměřená data, v tomto případě mračna bodů. Následně z těchto mračen extrahovat důležité objekty (charakteristiky) a implementovat algoritmus na registraci a skládání mapy. Výsledky potom demonstrovat na příkladech a příslušně je vyhodnotit.

První kapitola se věnuje seznámení s problematikou SLAM, s používanými algoritmy a možnostmi jejího využití. Další kapitola představuje několik druhů senzorů LIDAR, kde je také popsán používaný senzor k této diplomové práci. Následující kapitoly se věnují programové implementaci a následnému měření a vyhodnocení různých modelových prostředí.

V textu práce se často vyskytují anglické názvy metod a algoritmů z důvodu návazností na užité reference a z důvodu absence vhodného překladu.

1. Rešerše problematiky SLAM a stručný popis zvolených metod

1.1. Úvod do problematiky SLAM

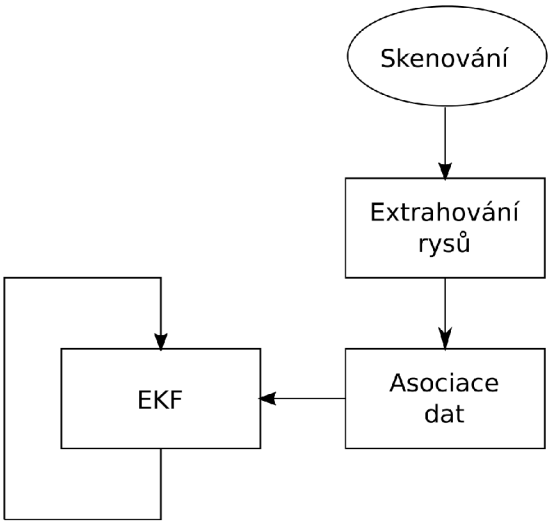
Pojem SLAM znamená "Simultaneous localization and mapping", to lze vlastními slovy vyložit jako současná tvorba mapy a vlastní orientace v ní. Je nespočet způsobů jak lze tuto techniku implementovat jak z pohledu hardware tak software.

SLAM se dá rozdělit do několika kroků [1]. První je sběr dat, data mohou být reprezentována různě s ohledem na použité zařízení. Nejčastěji se používají 2D nebo 3D laserové skenery (LIDAR), kamery, různé senzory vzdálenosti, sonary, atd. Není podmínkou používat pouze jeden senzor, často se kombinují, aby bylo dosaženo požadované přesnosti a dosahu. Při kombinaci senzorů se potom zpracovává tzv. fúze dat.

Po nasnímání dat následuje druhý krok, který má za úkol hledat významné objekty (charakteristiky) a po zpracování (identifikace, klasifikace) je uložit. V tomto bodě se z mračna bodů vyseparují důležité body, hrany atp. Tyto objekty jsou potřebné pro další krok a to konkrétně *Asociace dat*, která podle důležitých objektů ze dvou měření určí rozdíly v posunutí a natočení.

V poslední kroku se neustále přepočitává mapa (může být různě interpretována) na základě posledních měření. Úpravují se pozice charakteristik, pozice robota a zvyšuje se také důvěryhodnost znovu nalezených objektů. Často se v tomto kroku používá tzv. *EKF* (Extended Kalman filter).

Všechny uvedené kroky (obr. 1.1) se provádí několikrát za sekundu a je vyžadováno, aby zpracování probíhalo v reálnem čase.



Obrázek 1.1: Průběh SLAM.

Při aplikaci SLAM lze identifikovat některé třídy problémů, jejichž řešení není triviální. Jedna z nich jsou pohybující se objekty v měřeném prostoru. S těmi je potřeba počítat a také je připadně detekovat, protože jedna chyba může ohrozit použitelnost celé mapy. Další třídou je uzavření smyčky na výsledné mapě, tím se myslí případ, kdy se skenuje nějaká oblast a po určitém čase, se robot opět vrátí do už jednou měřeného prostředí. Poslední, dnes již ne až tak úplně aktuální, je výpočetní náročnost.

1.2. Extrahování důležitých rysů

Přesná lokalizace významných objektů je důležitým aspektem SLAM. Při každé nepřesnosti nebo chybě vzniká riziko, že při pozdějším zanášení objektů do mapy, může dojít k znehodnocení celého naměřeného souboru dat.

Do významných charakteristik se zahrnují všechna důležitá místa, která jsou potřebná pro pozdější orientaci. Patří tam rohy místností, stěny na které se dá napasovat nějaký geometrický útvar, atp. Pohybující objekty se také lokalizují, ne kvůli pozdější orientaci, ale aby nedošlo k chybnému přidání do mapy.

Existují dva způsoby jak se provádí detekce [2]. První je nalezení významných objektů v prvním měření a v dalším se pouze hledá v okolí daných souřadnic. Tento způsob se dá použít pouze v případech, kdy je jisté, že robot nezmění svou pozici o více než stanovené hraniční posunutí. Druhý způsob je hledání v obou měřeních nezávisle na sobě.

Extrahování charakteristik se skládá ze tří kroků:

- Detekce významných objektů,
- popis a vytváření vazeb mezi objekty,
- hledání stejných objektů v dalším měření.

1.2.1. Detekce významných objektů

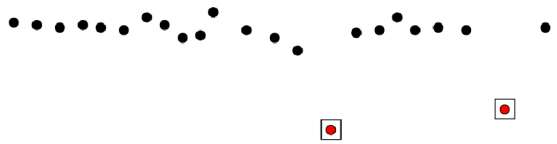
Detekce významných objektů se provádí několika způsoby. V konkrétním případě vždy závisí na implementaci s jakou reprezentací objektů algoritmus pracuje. Co se reprezentace týče, lze objekty reprezentovat jako body nebo přímky. Body mohou představovat například rohy nebo něčím významný bod. Přímky nejčastěji reprezentují hrany. Detekce se provádí v naměřených datech, tzv. *mračně bodů*, tím se myslí soubor naměřených dat většinou na jednu otáčku skeneru.

Algoritmus Spikes

Algoritmus *Spikes* [1] se používá pro hledání extrémů v mračně bodů. Hledají se takové body, které se významně liší. Příkladem může být stůl v měřené místnosti. Dokud algoritmus nenarazí na nohy stolu, jsou změny ve vzdálenosti bodů malé, v okamžiku kdy se

objeví body, které reprezentují nohy stolu je změna ve vzdálenosti bodů skoková. V algoritmu je potřeba si stanovit jak velká změna je považována za důležitou pro zapamatování.

Druhá možnost jak detekovat *spike* neboli hrot je pomocí tří hodnot. Mějme tři body A, B a C jdoucí za sebou. Vypočteme jejich vzdálenosti, například v euklidovské metrice $A - B, C - B$ a tyto dva výsledky sečteme. Podle velikosti výsledné hodnoty usoudíme jestli bod B je duležitý či nikoliv. Na obrázku 1.2 jsou naznačené body, které algoritmus označí jako *spike* (v čtverečku).



Obrázek 1.2: Příklad mračna bodů, v kterém algoritmus najde významné body.

Tento postup se používá pro detekci pevných předmětů v místnosti. Nevýhodou jsou pohybující se objekty, protože je algoritmus považuje za významné objekty a to vede k chybám.

Sliding window

Pro detekci rohů je využívaná technika označována jako *Sliding window* [3] nebo-li pohybující se okno. Algoritmus pracuje se třemi body z pohyblivého okna, mezi kterými se vytvoří dva vektory a vypočte se úhel mezi nimi (obr. 1.3). Pokud úhel splňuje stanovenou podmínku je považován za roh.

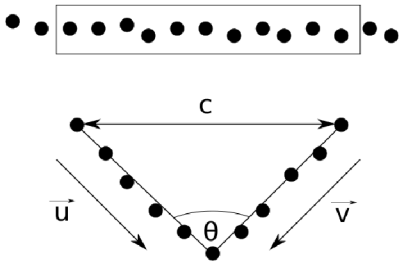
Velikost okna se volí mezi 11, 13 nebo 15 body, kde se považuje za vrchol 6., 7. nebo 8. bod. V případě 11 bodů se vytvoří dva vektory směřující do vrcholu (bod 6.) z obou konců. V tomto kroku je také důležite zkontrolovat vzdálenost c (obr. 1.3), která musí být větší než zvolené minimum. Vypočte se úhel, který svírají dané vektory a pokud je menší než 120° (není podmínkou), provádí se další zpětná kontrola.

Zpětná kontrola se provádí směrem k vrcholu. V každém kroku jsou uvažovány dva body blíže středu (2. a 10., 3. a 9., atd.) a provede se výpočet úhlu. Pokud ve všech krocích úhel splňuje stanovenou podmínku je považován za roh a ukládá se mezi významné objekty.

Pro výpočet úhlu θ se používá vzorec (1.1).

$$\theta = cos^{-1}\left(\frac{u \cdot v}{|u||v|}\right). \tag{1.1}$$

Při implementaci této metody je potřeba dávat pozor na detekci neexistujících rohů. Tyto body mohou narušit přesnost a správnost pozdějších kroků.



Obrázek 1.3: Body, na kterých se provádí detekce rohu.

Algoritmus Ransac

Algoritmus *Ransac* (Random sample consensus) [1] [4] je iterační algoritmus používaný pro odhad parametrů matematického modelu z naměřených dat. Obecně data obsahují body, které není možné přiřadit k modelu tzv. *outliers*, algoritmus je dokáže odhalit a nepracovat s nimi. Naopak data přiřaditelná modelu se označují jako *inlier*.

Ve spojitosti s SLAM se Ransac používá k detekci přímek v mračně bodů. Na začátku si algoritmus vybere náhodnou část naměřených bodů a pomocí metody nejmenších čtverců se pokusí data co nejlépe aproximovat. Následně se spočítá kolik bodů leží blízko této přímky a pokud je dosaženo stanovené meze, je možné zařadit tuto přímku mezi významné objekty.

Počet iterací se volí dostatečně vysoký, aby pravděpodobnost p byla nejčastěji $0,99$. Pravděpodobnost p představuje to, že alespoň jedno mračno bodů neobsahuje outlier. Potom u je pravděpodobnost, že kterýkoliv vybraný bod je inlier a opačně $v = 1 - u$ je pravděpodobnost nalezení otlieru.

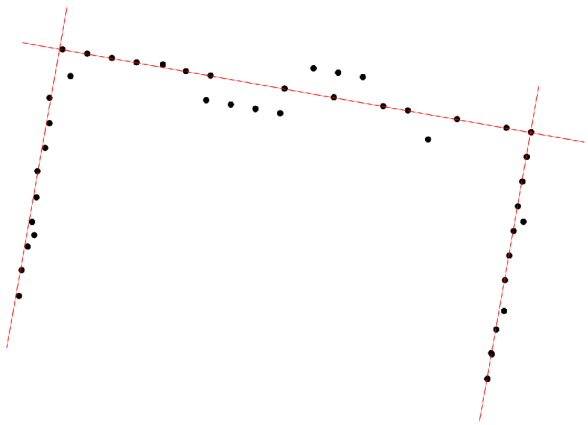
$$1 - p = (1 - u^m)^N, \tag{1.2}$$

kde m je minimální požadovaný počet bodů a N počet iterací kdy se těchto bodů dosáhne. Po vyjádření N se vzorec změní na (1.3).

$$N = \frac{log(1-p)}{log(1-(1-v)^m)}. \tag{1.3}$$

Na obrázku (obr. 1.4) je zobrazen výsledek algoritmu Ransac. Přímky znázorňují nejlepší aproximaci daných bodů se zvoleným minimálním počtem bodů.

Výhodou je přesnost aproximace dat, avšak nevýhodou je absence časových mezí. Časová náročnost se zvyšuje s počtem bodů. Data (mračna bodů) také nemohou obsahovat více než jeden model (topologie místnosti). V případě, že obsahují více modelů, algoritmus nenajde ani jeden.



Obrázek 1.4: Nalezené přímky pomocí algoritmu Ransac.

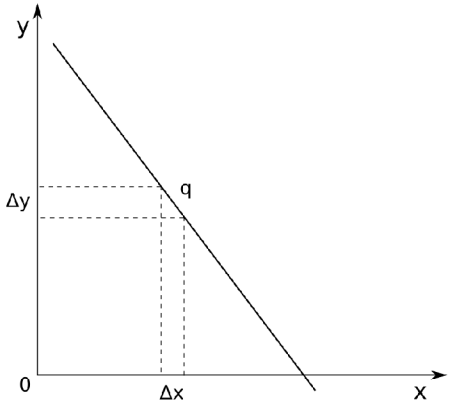
Split and Merge

Algoritmus *Split and Merge* [3] se používá pro hledání přímek v datech získaných ze skeneru. Předpokladem jsou body, v pořadí podle měření (rotace LIDARu). Algoritmus vychází ze směrnicové rovnice přímky (1.4).

$$y = kx + q, \tag{1.4}$$

kde $k = tg(\phi)$ je směrnice přímky, přičemž ϕ je orientovaný úhel s vrcholem v průsečíku přímky a první souřadnicové osy, q je tzv. úsek (vytnutý přímkou) na ose y, pokud $q = \infty$, tak je přímka rovnoběžná s osou y. Hodnotu q lze také vyjádřit jako podíl změny na ose y a x (1.5). Úsek q je zobrazen také na obr. 1.5

$$q = \frac{\Delta y}{\Delta x}. \tag{1.5}$$

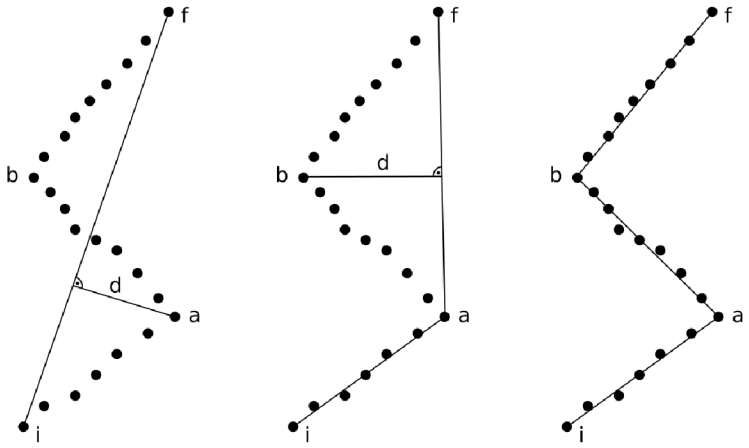


Obrázek 1.5: Úsek vytnutý přímkou na ose y.

Metoda používána v [3] pracuje tak, že hledá dělící body přímky, které mají maximální kolmou vzdálenost od přímky. Na začátku je přímka mezi prvním a posledním bodem měření $Ax + By + C = 0$, kde $A = y_f - y_i; B = x_f - x_i; C = -(By_f - Ax_f)$ (i je index prvního bodu a f je index posledního bodu). Potom pro všechny body mezi i a f se spočítá kolmá vzdálenost k přímce (1.6).

$$d_{\perp,k} = \frac{Ax_k + By_k + C}{\sqrt{A^2 + B^2}}. \tag{1.6}$$

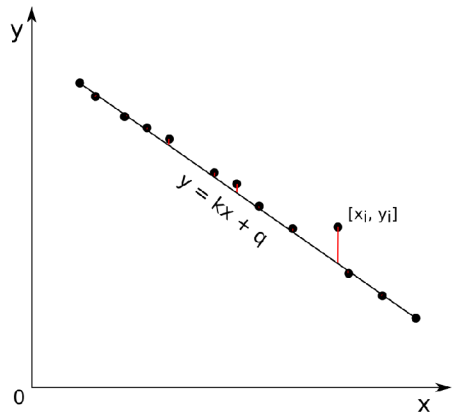
Tento postup se provádí rekurzivně do té doby než nalezené přímky dostatečně neaproximují naměřená data. Neboli v každé iteraci se spočítá kvadratická chyba a porovná se s ukončující podmínkou. V každé iteraci algoritmus hledá bod s největší kolmou vzdáleností, v tom místě následně rozdělí aktuální přímku, postup je vidět na obr. 1.6.



Obrázek 1.6: Průběh algoritmu Split and Merge.

Na obrázku 1.7 je přímka aproximující body s chybou, danou rovnicí (1.7). Chyba je součet kvadrátů vzdáleností vyznačených červeně.

$$S_E = \sum_{i=1}^{n} (y_i - (kx_i + q))^2. \tag{1.7}$$



Obrázek 1.7: Chyba při aproximaci přímky.

Houghova transformace

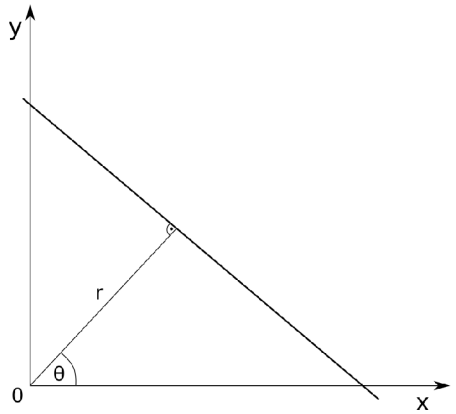
Houghova transformace [5] je algoritmus pro hledání významných charakteristik v obraze nebo mračně bodů. Používá se hlavně v analýze obrazu, počítačovém vidění a zpracování

digitálních obrázků. Hledat lze 2D charakteristiky (přímky, kružnice, atp.) a také 3D charakteristiky (plochy, válce, atp.).

V detekci přímek algoritmus nevychází z normálové reprezentace přímky podle rovnice (1.4), ale používá (1.8).

$$r = xcos\theta + ysin\theta, \tag{1.8}$$

kde r je vzdálenost od počátku k nejbližšímu bodu přímky a θ je úhel mezi osou x a spojnicí počátku s nejbližším bodem přímky.



Obrázek 1.8: Popis přímky při Houghově transformaci.

Hledá se přímka, která by se dala popsat rovnicí (1.8). Houghova transformace začína z počátečního bodu, který může být náhodně zvolený a prochází všechny body. Existuje tzv: *accumulator*, který je reprezentován maticí s řádky (r, θ). Accumulator v každém kroku počítá hodnoty r a θ z daného a ze sousedicího bodu. Následně se podívá do matice, jestli je už daná dvojice přidána, pokud né tak ji přidá, pokud ano tak pouze inkrementuje hodnotu výskytu.

Výsledkem algoritmu je matice s řádky (r, θ) a ke každé dvojici je přiřazena ještě hodnota počtu bodů, které na dané přímce leží. Podle zvoleného počtu minimálního výskytu dané dvojice se stanovují přímky.

Detekce kružnice se provádí obdobně. Velikost accumulatoru se rovná počtu všech bodů a hodnoty představují jestli se jedná o střed kružnice, hodnota a (jedna souřadnice středu) v rovnici (1.9).

$$(x - a)^2 + (y - b)^2 = r^2. \tag{1.9}$$

Pro všechny hodnoty a se hledá b, takové aby splňovalo předchozí rovnici.

1.2.2. Popis a vytváření vazeb mezi objekty

Po zpracování naměřených dat a nalezení významných objektů je potřeba je popsat. Popis je velmi důležitý, protože se používá k vytváření vazeb mezi robotem a nalezenými objekty a mezi objekty navzájem. K popisu se může používat matic [1] nebo např. indexované struktury.

Popis pomocí matic

Nejdůležitější je aktuální stav systému, který je reprezentován maticí **X** (1.10). První tři řádky matice obsahují informace o aktuální poloze robota, kde θ_r je jeho natočení. V dalších řádcích matice jsou aktuální pozice nalezených významných objektů s celkovým počtem n.

$$(X) = \begin{pmatrix} x_r \\ y_r \\ \theta_r \\ x_1 \\ y_1 \\ \dots \\ \dots \\ x_n \\ y_n \end{pmatrix} \tag{1.10}$$

Kovarianční matice **P** (1.11) popisuje vzájemný vztah všech nalezených objektů mezi sebou a robotem.

$$\mathbf{P} = \begin{pmatrix} \mathbf{A} & \mathbf{E} & \dots & \dots \\ \mathbf{D} & \mathbf{B} & \dots & \mathbf{G} \\ \dots & \dots & \dots & \dots \\ \dots & \mathbf{F} & \dots & \mathbf{C} \end{pmatrix} \tag{1.11}$$

Matice **P** se skládá z několika submatic. První je submatice **A**, která má tří řádky a tři sloupce a obsahuje kovarianci pozice robota. Další je matice **B** s dvěma řádky a sloupci představující kovarianci prvního nalezeného objektu. Obdobně potom matice **C** má stejnou velikost a je to kovariance posledního objektu.

Submatice **D** má dva řádky a tři sloupce a představuje kovarianci pozice robota a prvního objektu. **E** obsahuje kovarianci prvního objektu a pozice robota. Zbylé submatice (**G**, **F**), představují vzájemné kovariance mezi objekty.

Poslední prvek využívající se k popisu objektů je tzv. *The Kalman gain* **K** (1.12). Tato matice představuje míru důvěryhodnosti pozice nalezených a uložených objektů.

$$\mathbf{K} = \begin{pmatrix} x_r & x_b \\ y_r & y_b \\ t_r & t_b \\ x_{1,r} & x_{1,b} \\ y_{1,r} & y_{1,b} \\ \dots & \dots \\ x_{n,r} & x_{n,b} \\ y_{n,r} & y_{n,b} \end{pmatrix} \tag{1.12}$$

Hlavním významem matice je předcházet chybám, které vznikají špatným měřením nebo předpokladem. Uvedená matice (1.12) je úzce spjata s aktuálním stavem **X** (1.10)

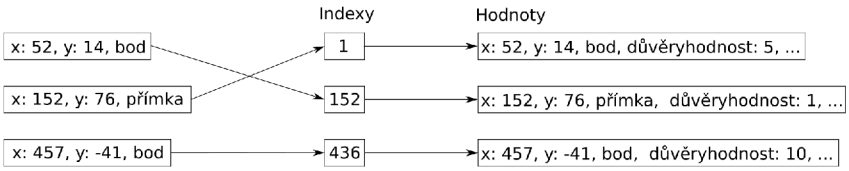
a pracuje následovně. Jsou uložené aktuální pozice významných objektů, dalším měřením se zjistí, že daný objekt je posunut o hodnotu Δ. Z dat se nezjistí, které měření je správné a tak je potřeba matice $\mathbf{K}$, která rozhoduje jak velká část z Δ se přičte k pozici objektu. Pokud míra věrohodnosti je malá tak se změna pozice projeví více a naopak pokud je vysoká, změna se téměř neprojeví.

První sloupec matice se vztahuje k věrohodnosti změn ve vzdálenosti a druhý sloupec k věrohodnosti změn v natočení. První tři řádky souvisí s pozicí robota a zbytek s pozicí objektů.

Popis pomocí indexovaných struktur

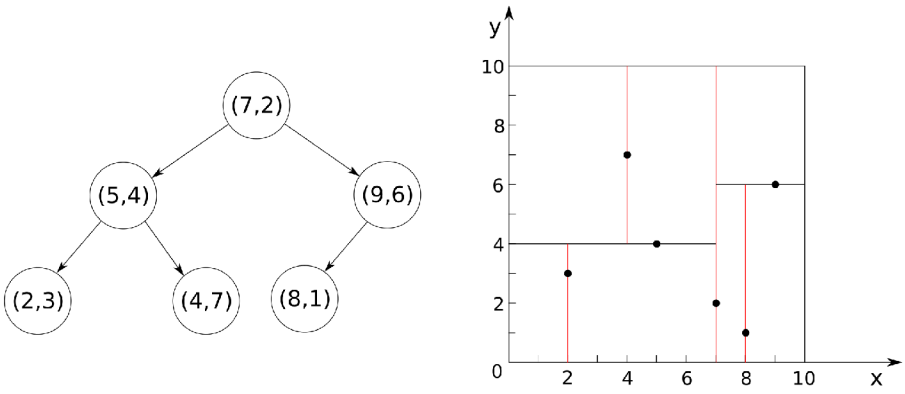
K popisu objektů a jejich vazeb není nutné používat předchozí reprezentaci pomocí matic. Využívá se také indexovaných struktur jako hešovací tabulky, hešování s ohledem na pozici, k-d stromy a další.

Podstatou *hešovacích tabulek* [6] (obr. 1.9) je ukládání dat do asociativního pole, kde indexy jsou vypočteny z ukládaných dat. Index se vypočte pomocí hešovací funkce, k tomu není potřeba používat celá data, ale pouze zvolené důležité části. Obdobně funguje i hešování s ohledem na pozici. Rozdílná je hešovací funkce, která je navržená tak, aby zvyšovala pravděpodobnost kolize u podobných prvků.



Obrázek 1.9: Příklad ukládání dat do hešovací tabulky.

K-d stromy [7] se používají pro ukládání objektů v k-dimenzionálním prostoru. Záměrem je rozložit prostor na malý počet buněk. Každá buňka by neměla obsahovat velký počet bodů. Výhodou je snadný a hlavně rychlý přístup k objektům na základě pozice. Body do stromu se přidávají na základě hledání mediánu, aby byl strom rovnoměrně zkonstruován (obr. 1.10).



Obrázek 1.10: Reprezentace bodů pomocí k-d stromu.

Výhodou popisu objektů pomocí indexovaných struktur je rychlý přístup k prvku bez prohledávání celého prostoru. Také se rychle hledají nejbližší objekty k daného bodu.

1.2.3. Hledání stejných objektů v dalším měření

Hledání stejných objektů v dalším měření je složité. Na jednu stranu většina nalezených objektů je stejných, ale na druhou stranu se nemusí detekovat všechny a také přibudou nové objekty. Proto nelze předpokládat, že počet a pořadí objektů bude stejné. Pro tyto předpoklady je potřeba přizpůsobit strategii hledání.

Při hledání objektů se definují nasledující vztahy [2], které udávají míru nalezení stejných prvků, kde TP je počet správně nalezených, FN je počet nenalezených, FP počet špatně nalezených, TN je počet špatně nalezených, které jsou odhaleny jako špatné.

Míra správně nalezených TPR:

$$TPR = \frac{TP}{TP + FN} = \frac{TP}{P}, \tag{1.13}$$

míra chybně nalezených objektů FPR:

$$FPR = \frac{FP}{FP + TN} = \frac{FP}{N}, \tag{1.14}$$

předvídané správně nalezené objekty PPV:

$$PPV = \frac{TP}{TP + FP} = \frac{TP}{P'}, \tag{1.15}$$

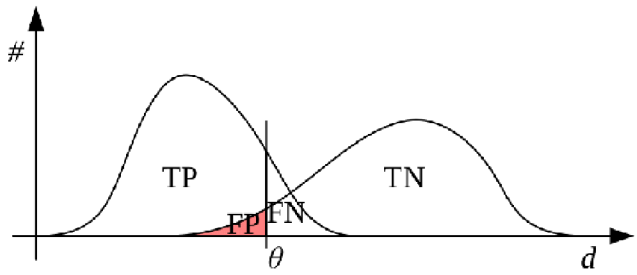
přesnost ACC:

$$ACC = \frac{TP + TN}{P + N}. \tag{1.16}$$

Pro danou strategii hledání je potřeba stanovit meze, ve kterých se bude hledání pohybovat. Vždy je nutné udělat kompromis, protože u požadavků s přehnanou přesností, mohou strategie selhat a nebude nalezen žádný objekt. Na obrázku 1.12 je zobrazena závislost správně a špatně nalezených objektů s ohledem na přesnost.

Nejjednodušší strategie prohledávání je projít všechny objekty a porovnat je se všemi nově nalezenými. Tento přístup není ovšem efektivní, protože se jedná o kvadratickou složitost a to je nežádoucí pro většinu aplikací.

Lepší strategie je hledání nejbližšího souseda, kde se prohledává pouze malé okolí. Nejlepší reprezentace pro tuto strategii je pomocí dříve zmíňovaných indexovaných struktur, protože každý soubor naměřených dat může obsahovat svoje datové struktury a hledání v nich je velice rychlé. Nevýhodou je, že hledání nejbližšího souseda je možné použít pouze v případech kdy změna pozice mezi měřeními je malá. Počítá se s malým okolím, které je přesnější a rychlejší k prohledání.



Obrázek 1.11: Poměr mezi správně nalezenými TP a špatně nalezenými TN s ohledem na přesnost θ [2].

Pokud je změna pozice velká je lepší použít jinou strategii. Například použít matici křižových korelací, z které vyplývá vztah mezi objekty. Na základě daných korelací se hledají objekty s podobnými vztahy v dalším měření. Existují také strategie, které obsahují učení a díky tomu jsou schopny predikovat pozici objektu.

1.3. Asociace dat

Obecně naměřená data v rozdílných krocích mohou mít jinou chybu měření a být různě deformována, proto nalezení vhodné transformace mezi měřeními, může být dosti složité. Využívané transformace se dají rozdělit na tuhé (rigid) a flexibilní (non-rigid). Tuhé transformace se skládají pouze z translace, rotace a škálování. Mezi nejjednodušší flexibilní transformace patří afinní transformace, ke které se ještě přidává anizotropní škálování a zkreslení. Flexibilní transformace se vyskytují v reálných problémech jako sledování pohybu, rozpoznávání tvaru a registrace obrazu v medicíně. Poslední zmiňová transformace

je také složitější, má vysoké množství transformačních parametrů a z toho důvodu bývá citlivější na šum a na body ležící mimo měřené okolí (outliers).

Tuhá transformace

Tuhé transformace [8] se skládají ze základních transformací, které se využívají v 2D i 3D počítačové grafice. Dále budou představeny transformace ve 2D. Nejjednodušší je translace (1.17), definována vektorem (pro jeden bod) nebo maticí (pro více bodů).

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}, \tag{1.17}$$

kde $(x'y')^T$ je bod po translaci z původní pozice $(xy)^T$. Mezi základní transformace patří také rotace (1.18), definována čtvercovou maticí $\mathbf{R}$.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}, \tag{1.18}$$

kde $(x'\ y')^T$ je bod po rotaci z původní pozice $(x\ y)^T$. Matice $\mathbf{R}$ představuje rotaci ve směru hodinových ručiček o úhel θ, v případě rotace proti směru hodinových ručiček se používá transponovaná matice $\mathbf{R}^T$. Další transformace je škálování (1.19), která pozici bodu vynásobí konstantou c, kde pro zmenšení je $c < 1$ a naopak pro zvětšení je $c > 1$.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = c \begin{pmatrix} x \\ y \end{pmatrix}. \tag{1.19}$$

Tyto tři uvedené transformace se řadí k tuhým a v algoritmech se většinou používají všechny najednou (1.20).

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = c \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}, \tag{1.20}$$

Flexibilní transformace

Do skupiny flexibilních patří v první řadě afinní transformace [9], která se skládá pouze z translace a lineární transformace. Lineární transformace ve 2D je obecně definována jako čtvercová matice a má tvar:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}, \tag{1.21}$$

v této matici jsou spojené elementární transformace, které lze rozdělit takto:

- změna měřítka:

$$\begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{1.22}$$

- rotace:

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{1.23}$$

- translace o vektor se souřadnicemi $(t_x\ t_y)^T$:

$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}, \tag{1.24}$$

Skládáním elementárních vznikají složitější transformace a obecně mají tvar (1.21). Transformace bodu $(x\ y)^T$ v afinním prostoru má tedy tvar:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{1.25}$$

V tomto kroku se k daným naměřeným mračnům bodů hledá vhodná transformace, aby se odvodilo skutečné natočení a posunutí robota. Existují různe přístupy, které využívají tuhé nebo flexibilní transformace. Používané algoritmy mají své výhody a nevýhody, které se vztahují ke konkretnímu problému. V SLAM aplikaci je potřeba, aby algoritmy byly velmi rychlé a robustní. Robustní v tom smyslu, že by měly být odolné vůči šumu a bodům, které nepatří k naměřeným datům tzv. *outliers*.

Dále je představeno několik algoritmů k zarovnávání mračen bodů. Všechny využívají jiných principů a mají různé vlastnosti.

1.3.1. ICP algoritmus

Algoritmus *ICP* (Iterative Closest Point) [10] se používá pro 2D, 3D rekonstrukci povrchů, k lokalizaci robota a také k planování optimální cesty robota. V algoritmu jsou dva mračna bodů. Jeden soubor dat se považuje za statický $\mathbf{X} = [x_1, \cdots, x_n]$ a druhý za pohyblivý $\mathbf{P} = [p_1, \cdots, p_n]$. Hledá se taková kombinace translace a rotace, aby byla minimální průměrná (střední) kvadratická chyba (1.26).

$$E(\mathbf{R}, \mathbf{t}) = \frac{1}{n} \sum_{i=1}^{n} ||x_i - \mathbf{R}p_i - \mathbf{t}||^2, \tag{1.26}$$

kde $\mathbf{R}$ je rotační matice, $\mathbf{t}$ translační vektor a n počet naměřených bodů. Před zahájením hledání vhodné transformace je ještě potřeba odečíst od všech bodů průměrnou hodnotu (1.27).

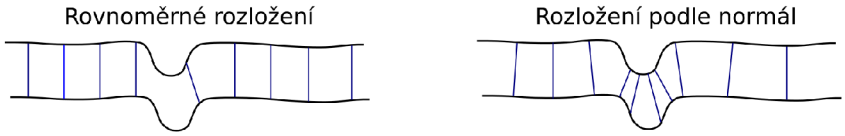
$$\mu_x = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad \mu_p = \frac{1}{n}\sum_{i=1}^{n} p_i. \tag{1.27}$$

Po odečtení se mračna bodů označují jako **X'** $= x_i - \mu_x = x'_i$ a **P'** $= p_i - \mu_p = p'_i$. Algoritmus je iterativní a konverguje rychleji, když inicializační transformace je dobře odhadnuta. ICP má různé vlastnosti jako rychlost, stabilita, tolerance, atd. Každá vlastnost je závislá na použité variantě.

Varianty algoritmu ICP:

- Výběr bodů,
- přiřazení váh shodám,
- asociace dat,
- neuvažování některých bodů (outliers).

První varianta *výběr bodů* má několik modifikací. Nejtrivialnější je pracovat se všemi body, další možnost je vybírat náhodně několik bodů ke zpracování. Vybírat se může také na základě důležitosti bodů (rohy, hrany, atd.) nebo na základě normál, kde je zajištěno, že vzorky mají normály rozložené rovnoměrně.



Obrázek 1.12: Způsoby rozložení bodů.

Pokud se využíva důležitých bodů tak se zvýší efektivita a přesnost, ale je zapotřebí preprocessing.

Další variantou je přiřazení váh shodám, je to metoda, při které se body ohodnotí váhami dle důležitosti. Tato strategie je velmi závislá na naměřených datech, protože je potřeba zaručit, že data obsahují dostatek důležitých bodů. Také je potřeba najít rozumnou mez jak hledat důležité body s ohledem na využitý čas.

Asociace dat má nejvyšší efekt na konvergenci a rychlost. Probíha na základě spojování určitých bodů, jak už nejbližších, nebo ležících na normále. K hledání nejbližších bodů se často používá struktur jako k-d stromy.

ICP je výhodný k počítání transformací mezi naměřenými daty. Nejdůležitější je správně detekovat důležité objekty.

Algoritmus se používá pro různe účely. Například se dá použít při naměření části dat k jejich úplné rekonstrukci [11]. Pokud je k dispozici databáze objektů, tak v ní tento algoritmus dokáže hledat a najít objekt, kterému patří naměřená část.

1.3.2. Statistická metoda k zarovnávání objektů s využitím afinní transformace

Tato metoda [12] rozkládá afinní matici (1.21) na další tři matice, které lze odhadovat každou zvlášť. Dvě matice jsou získány pomocí Choleského rozkladu a poslední pomocí třetího centralního momentu mračna bodů.

Třetí centrální momenty nejsou známy přímo, ale lze je odhadnout pomocí vzorce (1.28), kde k-tá souřadnice bodu se vyjádří takto $x_k = [x_{1,k}\ x_{2,k}]$ a průměrná hodnota $\bar{x}$ se dá vyjádřit dle vzorce (1.29).

$$\hat{c}_{ij} = \frac{1}{N}\sum_{k=1}^{N}\left(x_{1,k} - \bar{x}_1\right)^i \left(x_{2,k} - \bar{x}_2\right)^j, \tag{1.28}$$

$$\bar{x} = \left(\bar{x}_1, \bar{x}_2\right)^T = \frac{1}{N}\sum_{k=1}^{N} x_k, \tag{1.29}$$

Dalším krokem je z kovarianční matice $\mathbf{C}$ a $\mathbf{C}'$ (pro druhé mračno bodů) získat pomocí Choleského rozkladu matice $\mathbf{F}$ a $\mathbf{F}$'. Na naměřená mračna bodů $\mathbf{X}$ a $\mathbf{X}'$ se ještě aplikuje *whitening* transformace k získání $\mathbf{Y}$ a $\mathbf{Y}'$. *Whitening* transformuje vektor náhodné veličiny spolu s kovarianční maticí do nového vektoru, který má kovarianční matici $\mathbf{I}$ (jednotková matice). Dále se algoritmus dělí na dva způsoby, kde záleží na předpokladu jestli se jedná pouze o rotaci, nebo o rotaci a zrcadlení. Pro daný předpoklad se spočítájí náklady pro zvolený úhel α dle vzorce (1.30).

$$J(\alpha) = ||c' - \mathbf{B}\boldsymbol{\theta}||, \tag{1.30}$$

kde $\mathbf{c}'$ je vektor (1.32), $\mathbf{B}$ čtvercová matice a $\boldsymbol{\theta}$ vektor závislý na úhlu α.

$$\boldsymbol{\theta} = \begin{pmatrix}\cos^3(\alpha) & \cos^2(\alpha)\sin(\alpha) & \cos(\alpha)\sin^2(\alpha) & \sin^3(\alpha)\end{pmatrix}^T, \tag{1.31}$$

$$\mathbf{c}' = \left(c'_{30}\ c'_{21}\ c'_{12}\ c'_{03}\right)^T, \tag{1.32}$$

pro který se složky vypočítají z rovnic (1.33a) až (1.33d).

$$c'_{30} = a_1^3 c_{30} + 3a_1^2 a_2 c_{21} + 3a_1 a_2^2 c_{12} + a_2^3 c_{03}, \tag{1.33a}$$
$$c'_{21} = a_3 a_1^2 c_{30} + 2a_3 a_1 a_2 c_{21} + a_3 a_2^2 c_{12} + a_4 a_1^2 c_{21} + 2a_4 a_1 a_2 c_{12} + a_4 a_2^2 c_{03}, \tag{1.33b}$$
$$c'_{12} = a_1 a_3^2 c_{30} + 2a_1 a_3 a_4 c_{21} + a_1 a_4^2 c_{12} + a_2 a_3^2 c_{21} + 2a_2 a_3 a_4 c_{12} + a_2 a_4^2 c_{03}, \tag{1.33c}$$
$$c'_{03} = a_3^3 c_{30} + 3a_3^2 a_4 c_{21} + 3a_3 a_4^2 c_{12} + a_4^3 c_{03}. \tag{1.33d}$$

Výpočet matice $\mathbf{B}$ závisí na předpokladu rotace nebo rotace a zrcadlení. Pro první předpoklad se matice $\mathbf{B}$ určí dle (1.34) a pro druhý předpoklad (1.35).

$$\mathbf{B} = \begin{pmatrix} c_{30} & 3c_{21} & 3c_{12} & c_{03} \\ c_{21} & 2c_{12} - c_{30} & c_{03} - 2c_{21} & -c_{12} \\ c_{12} & c_{03} - 2c_{21} & c_{30} - 2c_{12} & c_{21} \\ c_{03} & -3c_{12} & 3c_{21} & -c_{30} \end{pmatrix}, \tag{1.34}$$

$$\mathbf{B} = \begin{pmatrix} -c_{30} & 3c_{21} & -3c_{12} & c_{03} \\ c_{21} & -2c_{12} + c_{30} & c_{03} - 2c_{21} & c_{12} \\ -c_{12} & c_{03} - 2c_{21} & -c_{30} + 2c_{12} & c_{21} \\ c_{03} & 3c_{12} & 3c_{21} & c_{30} \end{pmatrix}. \tag{1.35}$$

Algoritmus je založený na minimalizaci nákladů (1.30) dle zvoleného α. Po nalezení minimálního úhlu se vypočítají hodnoty translace $\mathbf{t}$ (1.37) a afinní matice $\mathbf{A}$ (1.36). Kde matice a vektory označené apostrofem ($\mathbf{F}$', $\bar{\mathbf{x}}'$) náleží k druhému mračnu bodů.

$$\mathbf{A} = \mathbf{F}'\mathbf{R}\mathbf{F}^{-1}, \tag{1.36}$$

$$\mathbf{t} = \bar{\mathbf{x}}' - \mathbf{A}\bar{\mathbf{x}}. \tag{1.37}$$

Zarovnávání mračen bodů tímto způsobem je velice rychlé, avšak nevýhodou tohoto přístupu jsou centrální momenty, které jsou velice náchylné na šum a tím pádem i na *outpliers*. Takže tento algoritmus není použitelný pro SLAM bez speciální přípravy dat, která by smazala nevhodné body z naměřených dat.

1.3.3. Afinní registrace pro 2D mračna bodů

Algoritmus [13] potřebuje dva mračna bodů $\mathbf{P} = [p_1, \cdots, p_k]$ a $\mathbf{Q} = [q_1, \cdots, q_k]$, která mají stejný počet bodů k. Předpokládá se, že existuje permutace π, taková že:

$$q_{\pi(i)} = \mathbf{A}p_i + \mathbf{t}, \tag{1.38}$$

kde $\mathbf{A}$ je afinní matice a $\mathbf{t}$ je translační vektor. Dále je potřeba vypočítat středy mračen bodů m_p a m_q, které jsou potřebné k odvození translačního vektoru (1.39).

$$\mathbf{t} = m_q - \mathbf{A}m_p, \tag{1.39}$$

Z naměřených dat $\mathbf{P}$ a $\mathbf{Q}$ se určí kovarianční matice $\mathbf{S}_p$ a $\mathbf{S}_q$, které jsou mírou toho, jak se dvě veličiny měni vzájemně.

$$\mathbf{S}_p = \sum_{i=1}^{k} p_i p_i^T. \tag{1.40}$$

Dalším krokem je transformovat data $\mathbf{P}$ a $\mathbf{Q}$ na normované veličiny, které jsou relativní (bezrozměrné) a mají společnou neznámou rotační matici $\mathbf{R}$. Nyní za předpokladu, že p_i a q_i jsou komplexní čísla, je potřeba nalézt nejmenší stupeň elementárního symetrického polynomu, pro který platí $n \geq 3$ (1.41).

$$a_n = \Pi_n(p_1, \cdots, p_k) \neq 0. \tag{1.41}$$

Obdobně se určí b_n pro druhé mračno bodů. Vydělením polynomů $z^n = b_n/a_n$, vznikne polynom, z kterého jeden z n kořenů bude obsahovat požadovanou rotační matici $\mathbf{R}$. Pro každý kořen se vyjádří možnost θ:

$$\theta = \tan^{-1}\left(\frac{y}{x}\right) + \frac{2n\pi}{d}, \tag{1.42}$$

kde $b_d/a_d = x + iy$, pro všechna $n = 0, 1, \cdots, d-1$ a d je zvolená možnost. Po získání rotační matice $\mathbf{R}$ lze vypočítat afinní matici $\mathbf{A}$:

$$\mathbf{A} = \mathbf{S}_q^{\frac{1}{2}} \mathbf{R} \mathbf{S}_p^{-\frac{1}{2}}. \tag{1.43}$$

Výhodou tohoto algoritmu je, že pokud jsou data bez šumu, tak vypočítá přesnou afinní transformaci. Také není potřeba, žádné předzpracování dat. Nevýhodou je rychlost algoritmu, protože každá iterace pro procesor s rychlostí 1,5 GHz trvá 5 až 7 sekund. Při porovnání s jinými algoritmy je několikanásobně pomalejší.

1.3.4. Zarovnávání pomocí souvislého posunu bodů

V algoritmu [14] jsou naměřeny dva soubory bodů $\mathbf{Y}$ a $\mathbf{X}$, první z nich je považován za *GMM* (Gaussian Mixture Model) a druhý za body generovány z *GMM*. Zarovnávání mračen se provádí na základě odhadu hustoty pravděpodobnosti, kdy pro optimum jsou data zarovnána. Hlavní myšlenkou je souvislý pohyb jednoho mračna bodů dokud nedojde k zarovnání. Jedná se o EM (expectation–maximization) algoritmus, který má tři modifikace pro:

- Tuhou transformaci,
- flexibilní transformaci,
- rychlou transformaci.

Tuhá transformace

Na začátku se inicializují hodnoty $\mathbf{R} = I, t = 0, s = 1$ a $0 \leq w \leq 1$. $\mathbf{R}$ je rotační matice, $\mathbf{t}$ translační vektor, s měřítko a w je parametr, který symbolizuje odhad jak velký šum se vyskytuje v naměřených datech. Také je potřeba spočítat rozptyl σ^2 podle vzorce (1.44).

$$\sigma^2 = \frac{1}{DNM} \sum_{n=1}^{N} \sum_{m=1}^{M} ||x_n - y_m||^2, \tag{1.44}$$

kde D je dimenze (pro 2D $D = 2$), N je počet bodů v $\mathbf{X}$ a obdobně M je počet bodů v $\mathbf{Y}$. Dále následuje EM část algoritmu, kdy v kroku E se spočítá rozdělení pravděpodobnosti $\mathbf{P}$ (1.45).

$$p_{mn} = \frac{\exp^{-\frac{1}{2\sigma^2}||x_n - (s\mathbf{R}y_m + \mathbf{t})||^2}}{\sum_{k=1}^{M} \exp^{-\frac{1}{2\sigma^2}||x_n - (s\mathbf{R}y_k + \mathbf{t})||^2} + (2\pi\sigma^2)^{D/2} \frac{w}{1-w} \frac{M}{N}}. \tag{1.45}$$

V kroku M se počítá $\mathbf{R}$, $s, \mathbf{t}$ a σ^2. $\mathbf{R}$ se vypočítá podle (1.47). Je potřeba si připravit některé hodnoty (1.46a) až (1.46c), kde $\mathbf{1}$ je sloupcový vektor samých jedniček.

$$N_P = \mathbf{1}^T \mathbf{P} 1, \qquad \mu_x = \frac{1}{N_P} \mathbf{X}^T \mathbf{P}^T \mathbf{1}, \qquad \mu_y = \frac{1}{N_P} \mathbf{Y}^T \mathbf{P} \mathbf{1}, \tag{1.46a}$$

$$\hat{\mathbf{X}} = \mathbf{X} - \mathbf{1}\mu_x^T, \qquad \hat{\mathbf{Y}} = \mathbf{Y} - \mathbf{1}\mu_y^T, \tag{1.46b}$$

$$\mathbf{A} = \hat{\mathbf{X}}^T \mathbf{P}^T \hat{\mathbf{Y}} \tag{1.46c}$$

V matici $\mathbf{R}$ (1.47) jsou hodnoty $\mathbf{U}$, $\mathbf{V}$ z singulárního rozkladu matice $\mathbf{A}$ a $\mathbf{C}$ je diagonální matice obsahující na diagonále $(1, \cdots, 1, det(\mathbf{U}\mathbf{V}^T))$.

$$\mathbf{R} = \mathbf{U}\mathbf{C}\mathbf{V}^T. \tag{1.47}$$

Také je potřeba v každém kroku spočítat měřítko s dle (1.48), translační vektor $\mathbf{t}$ (1.49) a σ^2 (1.50). V těchto vzorcích se vyskytuje operátor $tr()$, který se označuje jako stopa (trace) matice a vypočítá se jako součet prvků na diagonále.

$$s = \frac{tr(\mathbf{A}^T\mathbf{R})}{tr(\hat{\mathbf{Y}}^T d(\mathbf{P}\mathbf{1})\hat{\mathbf{Y}})}, \tag{1.48}$$

$$\mathbf{t} = \mu_x - s\mathbf{R}\mu_y, \tag{1.49}$$

$$\sigma^2 = \frac{1}{N_P D}(tr(\hat{\mathbf{X}}^T d(\mathbf{P}^T 1)\hat{\mathbf{X}}) - s\ tr(\mathbf{A}^T\mathbf{R})). \tag{1.50}$$

EM část algoritmu se provádí, dokud ho nezastaví nějaká ukončovací podmínka. Podmínka může být ve tvaru: maximální počet iterací, dostatečně malá σ^2 nebo pokud nedochází ke zlepšení. Zarovnání druhého mračna bodů k prvnímu se potom provádí dle vzorce $\tau(\mathbf{Y}) = s\mathbf{Y}\mathbf{R}^T + \mathbf{1}\mathbf{t}^T$.

Flexibilní transformace

Algoritmus pro flexibilní transformaci, konkrétně pro afinní, je obdobný jako pro tuhou, akorát se místo rotační matice $\mathbf{R}$ používa afinní matice $\mathbf{B}$ a neuvažuje se měřítko s. Po vypočítání $\mathbf{B}$ se opět určí translační vektor $\mathbf{t}$ a rozptyl σ^2.

$$\mathbf{B} = (\hat{\mathbf{X}}^T\mathbf{P}^T\hat{\mathbf{Y}})(\hat{\mathbf{Y}}^T d(\mathbf{P}\mathbf{1})\hat{\mathbf{Y}})^{-1}, \tag{1.51}$$

$$\mathbf{t} = \mu_x - \mathbf{B}\mu_y, \tag{1.52}$$

$$\sigma^2 = \frac{1}{N_P D}(tr(\hat{\mathbf{X}}^T d(\mathbf{P}^T\mathbf{1})\hat{\mathbf{X}}) - tr(\hat{\mathbf{X}}^T\mathbf{P}^T\hat{\mathbf{Y}}\mathbf{B}^T)). \tag{1.53}$$

Ukončující podmínky jsou stejné jak u modifikace pro tuhé transformace, akorát výsledná transformace se spočítá jako $\tau(\mathbf{Y}) = \mathbf{Y}\mathbf{B}^T + \mathbf{1}\mathbf{t}^T$. Zdroj [14] také obsahuje metodu pro flexibilní transformace, která je založená na dislokační funkci $\tau(\mathbf{Y}, v) = \mathbf{Y} + v(\mathbf{Y})$.

Rychlá transformace

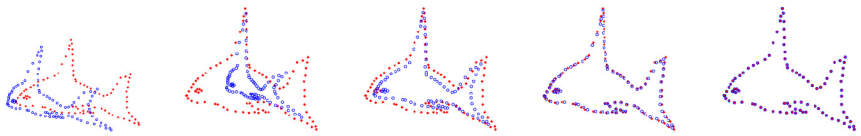
Při použití dříve zmiňovaných možností, je velice časově náročný výpočet matice rozdělení pravděpodobnosti $\mathbf{P}$ a ta je potřeba v každé iteraci. Z toho důvodu je zde představen způsob rychlejšího výpočtu pomocí *rychlé Gaussovy transformace* (FGT). FGT je pro rychlý součet exponenciálů a zjednodušuje výpočetní složitost z $\mathcal{O}(MN)$ na $\mathcal{O}(M + N)$. Hlavní

myšlenkou Gaussovy transformace je rozšířit Gaussovy funkce pomocí zkrácených Hermitových polynomů a aproximovat až do dosažení požadované přesnosti. Pomocí této transformace lze matici **P** spočítat pomocí rovnice (1.54), kde a vychází z rovnice (1.55). Používaná matice **K** je získána z FGT.

$$\mathbf{P} = \mathbf{K}\ d(a), \tag{1.54}$$

$$a = 1./(\mathbf{K}^T\mathbf{1} + c\mathbf{1}). \tag{1.55}$$

Díky této transformaci je výpočet několika násobně rychlejší a lze použít v aplikaci SLAM. Pro lepší představivost jak tento souvislý posun bodů funguje je na obrázku 1.13 zobrazeno několik iterací. Na obrázku je vidět jak se celý soubor bodů škáluje v měřítku a zarovnává k druhému mračnu bodů, až je na konci přesně zarovnaný.



Obrázek 1.13: Výsledek souvislého posunu bodu po 0, 10, 20, 40, 50 iteracích [14].

Výhodou je zmiňovaná rychlost algoritmu, protože průběh jedné iterace je v milisekundách. Výpočet obsahuje nastavitelný parametr šumu a je použitelný pro více dimenzí. Nevýhodou je, pokud nová naměřená data obsahují velké množství nových bodů, tj. pokud je velký rozdíl v pozici robota, tak se algoritmus stává nestabilní a někdy úplně selže.

1.3.5. Registrace s použitím Gaussian Mixture Model

Registrace mračna bodů s použitím Gaussian Mixture Model [15] je založena na převodu diskrétních hodnot na spojitou pravděpodobnostní funkci (density function), která se jmenuje *Gaussian Mixture Model* (GMM). Výhodou takového převodu je zjednodušení algoritmu pro spojování mračen bodů. Obecně také mohou být body měřeny s rozdílnou vzorkovací frekvencí a to bývá problém pro běžné algoritmy, které předpokládají podobné rozložení bodů. Pravděpodobnostní funkce (Gaussian Mixture) má obecně tvar (1.56), kde funkce $\phi(x|\mu_i, \Sigma_i)$ je ze vztahu (1.57).

$$p(x) = \sum_{i=1}^{k} w_i \phi(x|\mu_i, \mathbf{\Sigma}_i), \tag{1.56}$$

$$\phi(x|\mu_i, \mathbf{\Sigma}_i) = \frac{\exp -\frac{1}{2}(x-\mu_i)^T \mathbf{\Sigma}_i^{-1}(x-\mu_i)}{\sqrt{(2\pi)^d |det(\mathbf{\Sigma}_i)|}}. \tag{1.57}$$

Mračna bodů jsou označena následovně, pro statický GMM model je to **S** a pro pohyblivý model **M**. Při každé iteraci algoritmu je potřeba kontrolovat jestli se zarovnávání zlepšilo. K měření se používá L2 vzdálenost dvou GMM modelů, kde se minimalizuje následující funkce (1.58).

$$d_{L2}(\mathbf{S}, \mathbf{M}, \theta) = \int (gmm(\mathbf{S}) - gmm(T(\mathbf{M}, \theta)))^2 dx, \tag{1.58}$$

kde $gmm(\mathbf{S})$ je Gaussian Mixture Model vytvořený z mračna bodů $\mathbf{S}$ a θ symbolizuje výslednou prostorovou transformaci (T) modelu $\mathbf{M}$. V algoritmu se používá funkce $f(\theta)$, která symbolizuje L2 vzdálenost mezi GMM. Tato funkce se dále optimalizuje pomocí quasi-Newton algoritmu a hledá se optimální transformace. Výpočet končí, když je splněno některé z ukončujících kriterií.

Výhodou je použitelnost pro 2D i 3D mračna bodů, také není podmínkou stejný počet naměřených bodů v modelech. Algoritmus je použitelný jak pro tuhé tak i flexibilní transformace.

1.4. EKF

EKF neboli *extended Kalman filter* [1] je algoritmus, který zpřesňuje polohu robota na základě nalezených objektů a senzorů pohybu. Algoritmus se dá rozdělit do tří kroků:

- Aktualizace stavu robota,
- úprava stavu robota ze znovu nalezených objektů,
- přidávání nových charakteristik (objektů).

První krok používá zpětnou vazbu ze senzorů pohybu. Z nich se vypočítají hodnoty změny v posunutí dx, dy a natočení $d\theta$. Aktualizace stavu probíhá tak, že se pouze přičte vypočtené hodnoty k aktuálnímu stavu. Tento krok změní pouze první tři řádky v matici stavu (1.10).

V druhém kroku se používá znovu nalezených objektů k úpravě stavu robota nebo pozice těchto objektů. Obvykle je aktuální stav robota i důležitých charakteristik zatížen nějakou chybou. Stav se mění na základě matice $\mathbf{K}$ (1.12), ve které jsou uloženy hodnoty důvěryhodnosti daných objektů a pozice robota. Díky této matici se zamezuje chybné úpravě stavu, protože nejvyšší váhu mají rysy, které byly nalezeny již vícekrát.

Poslední krok pouze přidává nové charakteristické rysy do báze objektů a tím i jejich pozici vůči robotu a dalším objektům.

Algoritmus EKF se nepoužívá přímo v této podobě. Většinou se modifikuje k použití za potřebných podmínek. V některých případech se první krok neprovádí, protože není použito takových senzorů pohybu, aby šlo jednoduše určit změnu pohybu. V takovýchto případech SLAM zpracovává pouze mračna bodů, nebo obrázky v případě kamer.

1.5. Třídy problémů implementace SLAM

Třídy problémů si budeme definovat jako stavy, na které je potřeba si dávat pozor a správně je zpracovat. První třídou je nalezení několika nerozeznatelných objektů blízko

u sebe. K rozeznání více objektů se používají filtry jako *JPDAF* (Joint Probability Data Association Filter) a *PHD* filter.

Další třídou je identifikace pohyblivých objektů, protože zařazením do báze objektů, může celý SLAM proces selhat. Důležité je takovéto objekty správně detekovat a správně zpracovat. K detekci pohyblivých objektů se používá *DATMO* (detection and tracking of moving objects) [16]. Tento algoritmus rozdělí běžný odhad v SLAM na dva samostatné odhady, kdy jeden se používá pro statické objekty a druhý pro pohyblivé. Rozdělení umožní používat SLAM s pohyblivými objekty v reálném čase.

Třetí třídou problémů je uzavření smyčky, neboli detekce znovu navštívených prostor a správna úprava váh již uložených objektů. V této části se používá algoritmus *SIFT* (scale-invariant feature transform), který dokáže také detekovat špatně nalezené body (outliers).

1.6. Zvolené metody

Metody jsou použity s ohledem na vlastnosti úlohy. Za prvé se nepoužívá žádných senzorů pohybu, které by umožnily určit pohyb robota bez složitého zpracování. Za druhé se používá pouze jeden senzor, RPLIDAR od firmy RoboPeak, a proto je potřeba se spoléhat na jeho přesnost a rychlost měření (vlastnosti skeneru jsou podrobně popsány v další kapitole).

K extrahování důležitých objektů se používají tyto metody: *Sliding Window* k detekci rohů a *Split and Merge* k detekci hran (přímek) v naměřených datech. V dalším kroku je použit algoritmus *Zarovnávání pomocí souvislého posunu bodů* s použitím *FGT*. Tato metoda byla zvolena z důvodu rychlosti a dobré funkce při testování na naměřených datech. Algoritmus souvislého posunu bodů má také několik nevýhod, které byly odhaleny při testování. Nevýhody a podrobnosti použití budou blíže popsány v kapitole *Programová implementace*.

2. Rešerše senzorů LIDAR a popis zvoleného řešení

Pojem LIDAR znamená "Light Detection and Ranging", jedná se o vzdálené snímání s využitím světla ve formě pulsů (laser). Pomocí laseru se měří vzdálenost od objektů tak, že je vyslán puls a měří se doba, za kterou se vrátí. Je zapotřebí, aby měření a vyhodnocování probíhalo velice rychle, protože rychlost světla je téměř 300 000 000 metrů za sekundu. Výpočet vzdálenosti d, kterou světlo urazilo rychlostí c za čas t vychází ze vzorce (2.1).

$$d = \frac{ct}{2}. \tag{2.1}$$

LIDAR senzor vysílá až 150 000 pulzů za sekundu v závislosti na nastavené frekvenci a přesnosti měřidla. Světlo může být ultrafialové, viditelné nebo infračervené v široké škále použití od letadel a satelitů, přes bezpečnost až po mapování.

Senzor může pořizovat 2D i 3D měření. Vyrábí se buď statické, které vyzařují pulzy v jednom směru, nebo pohyblivé. Pohyblivé jsou například rotační, které obsahují statický laser umístěný na rotující hlavě měřidla. Dále bude představeno několik různých LIDAR senzorů, spolu s jejich vlastnostmi.

Statické senzory

Jedná se prakticky o vnitřek pohyblivých senzorů. Tyto senzory se používají pro měření v jednom směru nebo k pohyblivému, za předpokladu připojení na motorek. LIDARy mají různé vlastnosti v závislosti na výrobci, například LIDAR-Lite v2 od PulsedLight (obr. 2.1) [17] měří od 0 do 40 m s přesností 1 cm. Dosahuje rychlosti až 500 měření za sekundu a v přepočtu vychází na 2730 Kč.

Pohyblivé senzory s omezeným úhlem měření

Pohyblivé senzory s omezeným úhlem měření se používají v aplikacích, kde není potřeba nebo nejde měřit celých 360°. Pro roboty, které nepotřebují vědět co se děje za nimi. Jako příklad kvalitnější LIDAR od firmy SICK [18] s dosahem 80 m a měřicím úhlem 190°. Skener má také technologii multi-echo, která dokáže vyhodnotit více odrazů z jednoho pulzu. To znamená, pokud paprsek prochází přes sklo, mlhu, déšť nebo prach, generuje v každém povrchu odraz. SICK LIDAR dokáže zpracovat až 5 těchto odrazů a vyhodnotit z toho reálnou vzdálenost. Cena měřidla se pohybuje kolem 207 000 Kč.

Dalším senzorem s omezeným úhlem měření je UXM-30LN-PW od firmy Hokuyo [19]. Tento LIDAR má dosah 30 m, měřicí úhel také 190° a je vybaven funkcí multi-echo. Navíc může detekovat objekty v nastavitelné zóně bezprostředně před skenerem. Cena se pohubyje kolem 104 000 Kč.

Obrázek 2.1: Statický LIDAR od firmy PulsedLight [17].

Obrázek 2.2: Vlevo LIDAR od firmy SICK [18] a vpravo od firmy HOKUYO [19].

Pohyblivé senzory s 360° úhlem měření

Pohyblivé senzory s měřeným úhlem 360° se skládají ze dvou části. První je hlava a druhá tělo. Ve většině případů, při měření, se hlava otáčí a na jednu otáčku odměří daný počet bodů (v závislosti na výrobci). V této kategorii je velice široký cenový rozsah v závislosti na parametrech. První dva představené skenery patří do levnějších variant.

První je sweep LIDAR od firmy Scanse (obr. 2.3 vlevo) [20] s dosahem až 40 m. Je velmi lehký, váha tohoto zařízení je 120 gramů a dokáže s rozlišením 1 cm změřit až 500 vzorků za sekundu. Cena LIDARu je v přepočtu 6 000 Kč. Další skener je RPLIDAR od firmy RoboPeak (obr. 2.3 vpravo) [21] s dosahem 6 m. Rozlišení ve vzdálenosti je menší než 0.5 mm a v měřeném úhlu je to méně než 1°. RPLIDAR je rychlejší a za jednu sekundu změří minimálně 2 000 vzorků. Cena tohoto skeneru je v přepočtu 9 000 Kč.

Obrázek 2.3: Vlevo sweep od firmy Scase [20] a vpravo RPLIDAR od firmy RoboPeak [21].

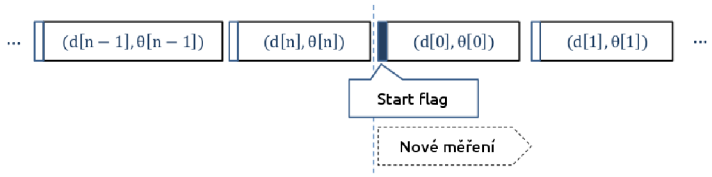
Třetí skener v této skupině je LD-OEM od firmy SICK (obr. 2.4 vlevo) [18]. LIDAR má dosah až 250 m (záleží na površích, od kterých se paprsek odrazí). Úhlové rozlišení se pohybuje od 0.125° a frekvence měření od 5 Hz do 20‘Hz. Cena tohoto skeneru se pohybuje okolo 150 000 Kč. Poslední představený skener je HDL-64E od firmy Velodyne (obr. 2.4 vpravo) [22]. Jedná se o nejpokročilejší senzor této firmy s dosahem 120 m, s možností měřit i vertikálně v rozsahu 26.8° a za sekundu dokáže naměřit 2.2 milionu bodů. Rozlišení je 0.08° a přesnost je vyšší jak 2 cm. Cena tohoto zařízení je přibližně 2 000 000 Kč.

Obrázek 2.4: Vlevo LD-OEM od firmy SICK [18] a vpravo HDL-64E od firmy Velodyne [22].

2.1. Parametry zvoleného senzoru

Pro tuto diplomovou práci byl zvolen RPLIDAR od firmy RoboPeak [21]. Skener má nastavitelnou frekvenci skenování, kdy při 5.5 Hz změří 360 vzorků. Maximální frekvence je 10 Hz. LIDAR se skládá ze dvou částí, skenovací část a část s motorem. Po napájení obou částí se hlava začne točit ve směru hodinových ručiček a měřit. RPLIDAR dokáže detekovat rychlost motoru a podle toho automaticky upravit frekvenci laseru. Skener má UART výstup pro přímé napojení na piny nebo USB výstup, při použití USB adaptéru.

Měření probíhá spojitě, kdy každý soubor dat začíná bitem *start flag* (obr. 2.5) a za ním pokračuje n naměřených dat. Naměřená data obsahují informace o vzdálenosti v mm, aktuální úhel ve stupních a kvalitu měření.



Obrázek 2.5: Spojité měření dat RPLIDAR.

2.1.1. Komunikační protokol RPLIDARu

Komunikace probíha ve formě paketů a inicializuje se vždy ze zařízení připojeného k LIDARu. Skener posílá data (odpověď) jenom na základě přijatého požadavku. Požadavky a odpovědí jsou trojího typu:

- Jeden požadavek, jedna odpověď,
- jeden požadavek, více odpovědí,
- jeden požadavek, žádná odpověď.

První mód se používá při požadavku zaslání nějakých dat (status skeneru, informace o skeneru, atd.). Software, který posílá požadavky, by měl vždy počkat na odpověď, protože další odeslané požadavky budou ignorovány.

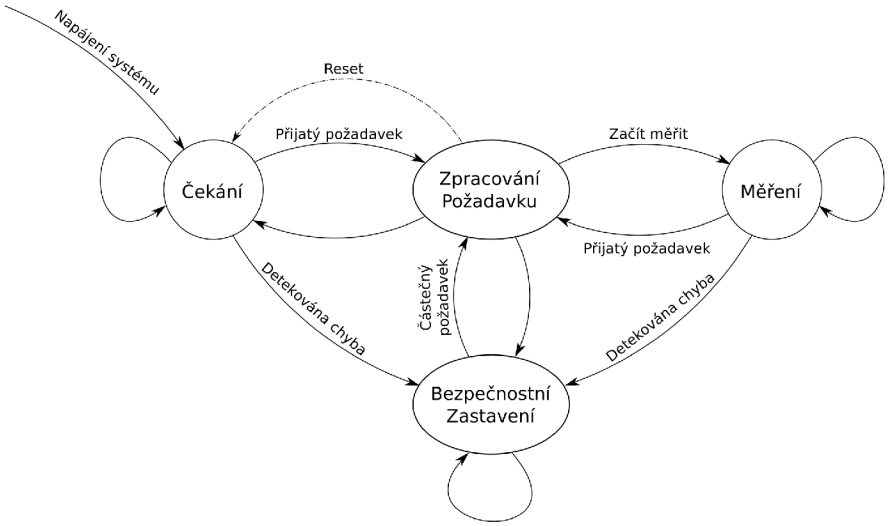
Druhý mód je pro měření. LIDARu se pošle požadavek, aby začal měřit, kde odpovědi jsou posílány spojitě. Pro zastavení měření je potřeba poslat *STOP* požadavek.

Požadavek pro zastavení měření je třetího typu, kde se nečeká na odpověď. Další takový příkaz je například *RESET*.

Paket s požadavky je definovaného tvaru, začíná opět start bitem, pokračuje 1 bajt s příkazem a nakonec obsahuje data. Data začínají 1 bajtem s informací o velikosti, dále obsahují samotná data a nakonec 1 bajt s kontrolní sumou.

Odpověď je definována nasledujícím způsobem. Na začátku obsahuje 2 bajty se startovacími příznaky, dále 30 bitů dat, 2 bity s označením typu a nakonec 1 bajt s označením typu dat.

RPLIDAR je obsluhován stavovým automatem (obr. 2.6) a má 4 hlavní stavy (čekání, měření, zpracování požadavku, zastavení z důvodu ochrany zařízení).



Obrázek 2.6: Vnitřní stavový automat RPLIDARu.

Do čekacího stavu se zařízení dostane automaticky po zapnutí napájení. Tady je systém v režimu úspory energie, dokud nepřijde požadavek na změnu. Ve stavu zpracovávání požadavku systém nepřijímá žádné další požadavky a po dokončení aktuálního pošle odpověď. Pokud se systém dostane do poruchového stavu, komunikace je pořád aktivní, ale nezpracuje požadavky na zahájení měření atp., dokud není odeslán *RESET* požadavek a chyba není odstraněná.

Se skenerem je dostupné i SDK, které obsahuje základní knihovny a ukázky kódu v C++. Software je open-source a multiplatformní, takže není problém s použitím na jakémkoliv operačním systému. Obsahuje také přednastavené workspaces pro Visual Studio od Microsoftu.

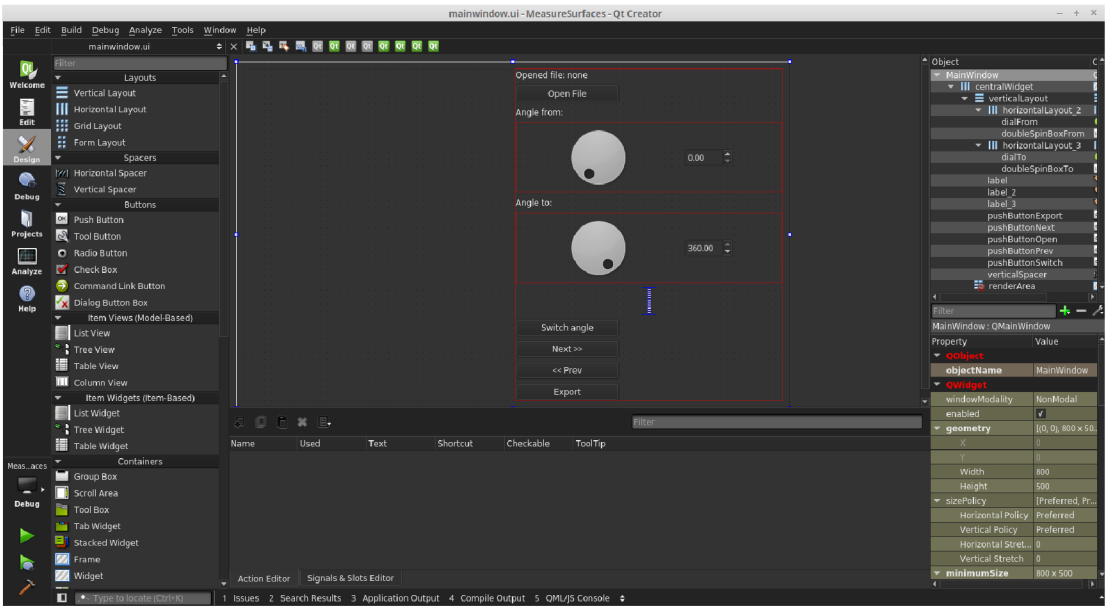
3. Programová implementace

3.1. Použité programové prostředí

Implementace je provedena v knihovně Qt [23]. Jedná se o multiplatformní knihovnu, která podporuje jak desktopové, tak i mobilní platformy. Výhodou je možnost vývoje aplikací s *GUI* (grafickým uživatelským rozhraním) v jazyce C++. Mezi podporované jazyky patří i Python, Ruby, C, Perl, Pascal a další. Qt obsahuje i knihovny pro práci s databazovými jazyky jako SQL a značkovacími jazyky XML. Dále podporuje prvky pro práci s grafikou a multimédii (mezi jinými i podpora pro OpenGL).

Aktuálně je Qt ve verzi 5.6 a je distribuována pod třemi různými licencemi. První je pro komerční vývoj aplikací, pro ty, kteří nechtějí sdílet svůj kód veřejně. Další je licence pro privátní použití (také studentské licence) a poslední je open source licence LGPL nebo GPL.

Pro vývoj aplikace se používá program Qt Creator (obr. 3.1), kde je možnost vytvářet aplikace s podporou Qt Widget, konzolové aplikace a aplikace Qt Quick, které jsou zaměřené na rychlý a snadný vývoj v jazyce QML. Novinkou od verze 5.5 je Qt Canvas3D aplikace, která obsahuje API podobné OpenGL a podporuje spoušťění 3D vykreslování pomocí příkazů v jazyce JavaScript. To umožňuje používat WebGL a dokonce sdílet kód aplikace mezi HTML a Qt Quick. Nicméně Qt Canvas3D je stále ve vývoji, a proto některé prvky ještě nejsou dostupné.

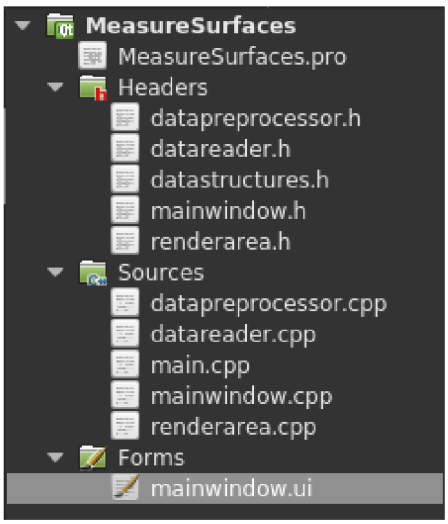


Obrázek 3.1: Vývoj GUI v aplikaci Qt Creator.

Diplomová práce je vytvořena jako Qt Widget aplikace, kde na obrázku 3.1 je vidět jak se vytváří grafické rozhraní. V Qt se GUI pro jedno okno jmenuje *formulář*. Formulář se nemusí vytvářet takto interaktivně, ale je možné také celé GUI sestavit pomocí C++. Do formuláře lze přidávat různé prvky (widgety), pro zarovnání prvků se používají rozložení (layouty). Jsou dostupné různé typy tlačítek, jako radio, check box, a další.

Pro organizaci výpisu se používají seznamy (výchozí, tabulkové, sloupcové, atd.). K zadávání hodnot od uživatele slouží textové prvky (Line Edit, Text Edit, atd.), prvky pouze pro numerické hodnoty (Spin Box, Double Spin Box, atd.) a ostatní k zadávání časových hodnot, datumů, a dalších.

Každý projekt se skládá z několika prvků (obr. 3.2). První a nejdůležitější je projektový soubor s příponou .pro. Ten obsahuje seznam všech položek, které se v projektu používají, jako například objekty, další knihovny, zdroje (obrázky, styly) a další. Projekt dále obsahuje hlavičkové soubory s deklaracemi, potom cpp sobory s definicemi objektů a nakonec formuláře pro GUI.



Obrázek 3.2: Soubory projektu rozděleny dle funkce.

Qt používá vlastní datové typy (pro složitější typy), aby byla zaručena multiplatformnost. Všechny typy začínají písmenem "q" nebo "Q" a patří tam například "qreal" pro prácí s reálnými čísly, "QString" pro práci s textovými řetězci a také vlastní typy jako bod (QPoint), přímka (QLine) a další.

Výhodou je možnost stylování GUI v jazyce QSS, které je téměř identický s CSS (kaskádové styly) umožňující měnit vzhled prvků. Prvky se dají selektovat podle typu nebo jména. Například pro selekci hlavního okna aplikace a pro změnu barvy pozadí na bílou, stačí napsat:

```
QMainWindow {
    background-color: #FFF;
}
```

Všechny selektory a všechny prvky, z kterých se daný widget skládá jsou v dokumentaci [23]. Pro práci s Qt v operačním systému Windows je dobré při instalaci si doinstalovat MinGW kompilátor, protože s kompilováním Qt aplikace pomocí kompilátoru od Visual Studia od Microsoftu, mohou nastat problémy.

3.2. Rozložení programu a základní funkcionalita

Aplikace je koncipována do jednoho okna, kde prvky (QTabWidget, QStackedWidget) umožňují měnit svůj obsah a není potřeba otevírat další okna. Hlavní okno je rozděleno do tří částí (obr. 3.3), kde 1. je hlavní nabídka aplikace. Při spuštění aplikace jsou dostupné tři záložky v hlavní nabídce: Soubor, Submapy a Nastavení, další se povolují dle stavu aplikace. Oblast označená jako 2. je k zobrazování obsahu, kde výchozí zobrazený prvek je vykreslování (mapy nebo jednoho měření). Poslední oblast (3.) obsahuje doplňující informace jako statistiky, počty submap, nastavení LIDARu a nahrané mapy.

Obrázek 3.3: Hlavní okno aplikace.

Nedostupné prvky v hlavní nabídce se zpřístupní po splnění jejich předpokladů. První záložka je *Statické vykreslování*, která je aktivní po nahrání dříve naměřených dat (možnost z jiné aplikace) v požadovaném formátu. Druhá nepřístupná záložka je *Vykreslování dat v reálném čase*, ta se zpřístupní po připojení ke skeneru. Aplikace je navržena tak, že je LIDAR připojen k serveru a komunikuje s aplikací pomocí LAN nebo WiFi. Hlavní důvod proč je LIDAR realizován jako samostatná jednotka, je kvůli použitelnosti. Aplikace SLAMu je výpočetně náročná a takové počítače/notebooky jsou rozměrově nevhodné k použití s autonomním vozidlem. Z toho důvodu bylo navrženo, že data se budou zpracovávat na notebooku, který bude komunikovat se serverem pomocí UDP protokolu a zpracovávat přijatá data.

Serverovou část tvoří LIDAR připojený k Raspberry Pi (miniaturní počítač). Raspberry Pi dle nastavených hodnot čte měření ze skeneru a v nastavených časových okamžicích

je odesílá. Detaily implementace komunikace a serverové části jsou popsány v několika dalších kapitolách.

3.3. Hlavní struktury v programu

V aplikaci se používá několik struktur pro efektivní uložení dat a práce s nimi. Snahou je mít všechna data uložená pouze jednou a vyhýbat se zbytečné duplicitě, protože to způsobuje těžko identifikovatelné chyby. Z toho důvodu jsou data předávaná pomocí reference nebo směrníku, aby byla dosažena vysoká rychlost a nevytvářely se zbytečné kopie.

Základní struktura je *data_t* (3.1), která představuje naměřená data bez zpracování.

$$\begin{pmatrix} \theta_1 & d_1 & q_1 \\ \theta_2 & d_2 & q_2 \\ \dots & \dots & \dots \\ \theta_n & d_n & q_n \end{pmatrix}, \tag{3.1}$$

kde každý řádek symbolizuje jeden naměřený vzorek a θ je úhel měření, d vzdálenost bodu, q kvalita měření a n počet naměřených dat na jednu otáčku LIDARu. Z toho vyplývá, že každý soubor naměřených hodnot z jedné otáčky je uložený v této struktuře.

Nejpoužívanější struktura je *area_t* (3.2), která obsahuje soubor naměřených dat z jedné otáčky. Obsahuje informace o počtu bodů n_p, matici přepočítaných hodnot na kartézské souřadnice $\mathbf{P} = (x_1,\ y_1;\ x_2,\ y_2;\ \cdots;\ x_{n_p},\ y_{n_p})$, distanční koeficient pro všechny body $\mathbf{K}$, nalezené důležité objekty jako rohy $\mathbf{C}$ a hrany $\mathbf{L}$ s jejich parametry $\mathbf{LP}$. Dále obsahuje data z registrace, jako možnost transformace *tran* (ano/ne), rotační matici $\mathbf{R}$, translační vektor $\mathbf{t}$ a informace o tom, která struktura (area_t) je předchozí *prev* a následující *next* (z pohledu registrace).

$$\left(n_p \quad \mathbf{P} \quad \mathbf{K} \quad n_c \quad \mathbf{C} \quad n_l \quad \mathbf{L} \quad \mathbf{LP} \quad \mathbf{R} \quad \mathbf{t} \quad \sigma^2 \quad s \quad tran \quad next \quad prev \right)^T. \tag{3.2}$$

Všechny hrany mají ještě strukturu s důležitými parametry, jako například hodnoty z rovnice přímky k, q a míru správné aproximace vyjádřenou jako chyba na bod. Je potřeba reprezentovat i pohyb robota, který je vyjádřen pomocí matice ukládající pozici. Také se ukládá vektor s indexy odkazujicími na první bod měření ve výsledné mapě (na které pozici začíná další oblast). U robota by se obecně měl ukládat i vektor určující směr, to ale v tomto případě není potřeba.

Nejdůležitější je reprezentace mapy, která je rozdělena na submapy (struktura *submap_t*) kvůli zrychlení prohledávání. V *submap_t* dochází ke kopírování bodů z *area_t*, protože je potřeba tyto body transformovat, aby bylo možné mapu složit v jeden souvislý celek. Každá submapa obsahuje pozice robota a důležitých objektů s ohodnocením.

3.4. Nahrávání naměřených dat

Nahrávání naměřených dat je dvojího typu. Rozlišují se nezpracovaná data, která svou reprezentací simulují měření z LIDARu a nahrávání uložených submap. Uložené submapy obsahují pouze důležité informace pro znovupoužití v aplikaci. Oba typy mají jiný formát a po nahrání se zpřístupňují další funkce.

3.4.1. Předem naměřená data

V případě předem naměřených dat je potřeba dodržet jejich formát (k správnému načtení). Data mohou být uložené například v textovém dokumentu, kde jedna otáčka skeneru je na jednom řádku a ten je zakončen značkou "-END-"a koncem řádku "\n". Měření mají následující formát (3.3), na začátku jsou dva podtržítka, kde první je v případě syncbitu nahrazeno za "S". Další údaje vždy začínají podtržítkem, kde $\_t : 1.34$ je hodnota úhlu ve stupních, $\_d : 01495.75$ je vzdálenost v mm a nakonec $\_q : 29$ je úroveň kvality signálu. Jedno měření je vždy zakončeno znakem "|".

$$\_\_t : 1.34\_d : 01495.75\_q : 29| \tag{3.3}$$

Předem naměřená data se nahrávají v první záložce (File) a po kliku na otevřít (Open) se otevře dialog pro výběr naměřených dat. Soubor s naměřenými daty musí být ve stejném adresáři jako aplikace, aby data byla úspěšně nahrána. Nahrávání probíhá ve třídě *DataReader*, která hodnoty načte do struktury *data_t* (3.1). Po načtení se data předají do třídy *DataPreProcessor*, kde probíhá předzpracování. Hodnoty se ukládají do struktury *area_t* (3.2), kde se provádí přepočet vzdálenosti d a úhlu θ na souřadnice x, y (3.4).

$$\begin{aligned} x &= d * \cos(\theta), \\ y &= d * \sin(\theta). \end{aligned} \tag{3.4}$$

Celý preprocessing se spouští ve vlastním vlákně, a proto nedochází k zamrzání programu nebo zastavování ze strany operačního systému. Pro práci s vlákny má Qt knihovnu *QtConcurrent*. Jsou tři možnosti jak spouštět asynchronní úlohy, první je *QtConcurrent::map()*, tato metoda aplikuje danou funkci na všechny prvky v daném poli. Další metoda je *QtConcurrent::filter()*, která smaže ty prvky z pole co neodpovídají dané filtrovací funkci. Poslední možnost je *QtConcurrent::run()*, která spustí funkci ve vlastním vlákně. První dvě metody jsou navrženy hlavně pro případy, kdy se pracuje se sdílenými daty. Se sdílenými daty lze pracovat i ve vlastním vlákně, ale je potřeba dodržet určité požadavky. V diplomové praci se pro práci s vlákny používá *QtConcurrent::run()*, spolu s třídou *QFutureWatcher*, která umožňuje monitorování vlákna. Díky monitorování, lze přesně určit v jakém stádiu je spuštěné vlákno (jestli už začalo, nebo bylo zrušeno, atd.). Velikou výhodou v Qt jsou signály a sloty, protože ty umožňují komunikaci mezi objekty. Pomocí funkce *connect()* může být *QFutureWatcher* spojen se signálem *finished()* a napojen na jakoukoliv požadovanou funkci. Při práci s vlákny, díky signálům, je vždy přesně známo, kdy dané vlákno zkončilo a může se pokračovat v navazujících krocích.

Aplikace je připravena na jakékoliv množství dat, protože měření (area_t) jsou uložena v dynamicky alokovaném poli. Dynamické pole je řešeno pomocí dvou hvězdičkového směrníku, a proto realokace je pouze předávání adres a z toho důvodu je velmi rychlá.

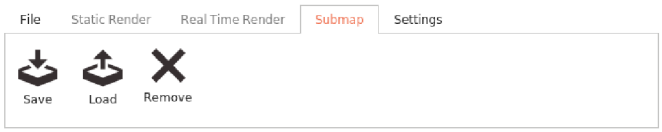
Po načtení dat se zobrazí statistiky (obr. 3.4). Pro tuto část jsou důležité pouze první dva údaje. První je počet měření, která se načetla a druhý jejich průměrná kvalita. Další statistiky se týkají vykreslování a registrace.

Obrázek 3.4: Dostupne statistiky po načtení měření.

3.4.2. Nahrávání uložených map

K načtení uložených map se používá záložka *Submap* (obr. 3.5) a tlačítko *Nahrát*. Pokud mapa byla uložena z této aplikace, má příponu ".smap". Tento soubor se rozděluje na několik částí:

- #points,
- #robot,
- #landmarks.

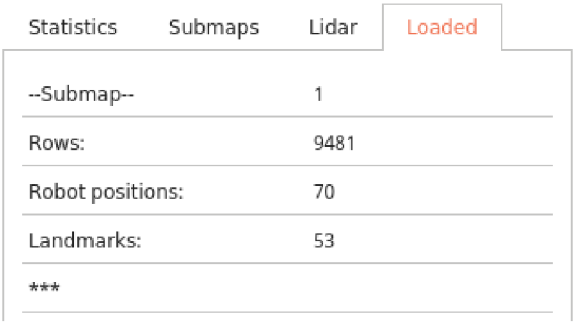


Obrázek 3.5: Nabídka pro načtení uložených map.

Každá část je na jednom řádku, kde bezprostředně řádek za ní je celkový počet prvků daného druhu, aby se urychlilo načítání. V případě bodů a pozice robota, následují řádky se souřadnicemi x, y oddělené čárkou. Z toho důvodu je potřeba ukládat data s desetinnou tečkou. V případě důležitých bodů (landmarks) (3.5) jsou první dvě hodnoty souřadnice x, y a dále údaj o věrohodnosti a typu. Čím vyšší věrohodnost objektu, tím byl nalezen víckrát. Typ v případě 0 se jedná o důležitý bod (roh) a 1 se jedná o hranu.

$$\begin{gathered} \#landmarks \\ 3 \\ -2110.33, -2760.47, 5, 0 \\ 563.693, 3974.48, 0, 0 \\ -5546.09, -3470.07, 0, 0 \end{gathered} \tag{3.5}$$

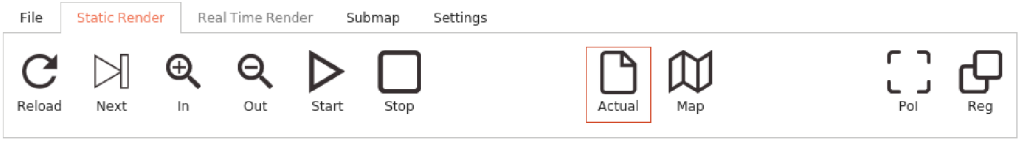
Po načtení se submapa zobrazí pod záložkou Nahrané (Loaded) v pravém sloupci (obr. 3.6). Každý výpis obsahuje informace o tom, kolik jakých objektů mapa obsahuje a po kliknutí na jakýkoliv řádek, se daná mapa vykreslí. Je možno načíst libovolné množství submap.



Obrázek 3.6: Seznam načtených map.

3.5. Zpracování statických dat

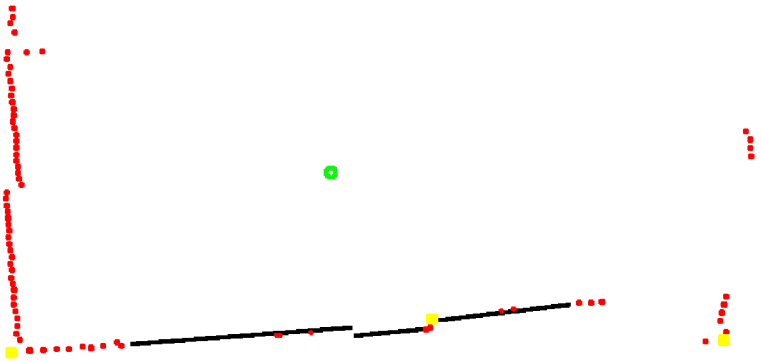
Po načtení předem naměřených dat se záložka statického vykreslování stane aktivní. Statické vykreslování má dva režimy. První režim je zobrazení dat po jednom měření s možností zobrazení důležitých objektů. Druhá možnost je spuštění registrace a s tím spojené stavby mapy. Na následujícím obrázku (3.7) jsou zobrazeny dostupné možnosti při statickém vykreslování.



Obrázek 3.7: Možnosti při statickém vykreslování.

První tlačítko *Načíst* (Reload) je výchozí a slouží k načtení prvního měření z dat. Po načtení se zpřístupní další funkce. Jako výchozí je zvoleno vykreslování samostatného měření bez dalších hodnot. K této funkci slouží tlačítka *Další* (Next), *Přiblížení* (In) a *Oddálení* (Out), pomocí, kterých si lze prohlédnout naměřená data. Pro znovu načtení prvního meření, stačí opět stistknout *Načíst*. Při vykreslování lze zobrazit nalezené důležité objekty v aktuálním měření tlačítkem *Pol* (obr. 3.8). Charakteristické rysy mající charakter rohů a bodů jsou zobrazeny žlutým bodem a hrany jsou vykresleny pomocí zelených přímek.

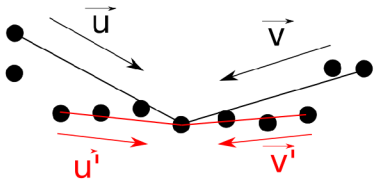
K registraci mračen bodů, stačí kliknout na *Reg* a následují dvě možnosti jak pokračovat. První je zobrazení po krocích, kde stačí klikat na tlačítko *Další*. Druhá možnost je zapnout automatickou registraci tlačítkem *Start*. Při zapnuté registraci se už důležité body hledají automaticky, proto tato možnost není dostupná.



Obrázek 3.8: Zobrazení důležitých bodů při statickém vykreslování.

3.5.1. Hledání důležitých objektů

K hledání rohů je použit algoritmus klouzavého okna (Sliding Window). Za roh se považuje ten bod, kde určující vektory svírají maximálně 143° a minimálně 40°. Tyto hodnoty byly určeny experimentálně a zvoleny proto, že měly nejlepší účinnost. Pokud se najde takovýto bod, je ještě prověřován jestli se opravdu jedná o roh. Prověřování je nastaveno na zkontrolování úhlu mezi vektory začínajícími v kontrolovaném bodě a končícími postupně v 1, 2, 3 a 4 bodě, v případě prvního vektoru se jedná o body předcházející kontrolovanému a u druhého vektoru zase naopak, se jedná o body následující po kontrolovaném bodě. Na obrázku 3.9 je případ, kde zpětná kontrola najde bod v kterém byl předpokládán roh chybně (černé vektory - roh, červené vektory - jeden krok zpětné kontroly).

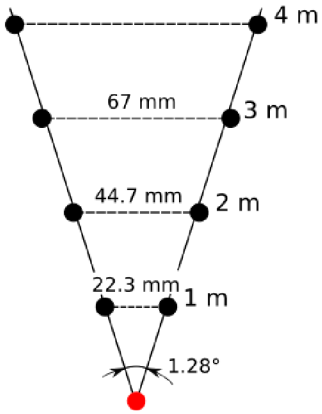


Obrázek 3.9: Zpětná kontrola u algoritmu klouzavého okna.

Jestli se jedná o roh nezáleží pouze na úhlu, protože by byly nalezeny také falešné rohy na místech, kde je měření nespojité (z nějakého důvodu chybí body). Proto je potřeba detekovat i velké změny v měřených bodech, nevýhodou však je, že vzdálenost bodů závisí na vzdálenosti od LIDARu, a proto byl zavedel distanční koeficient. Tento koeficient se počítá už v preprocesingu a v podstatě se jedná o rovnici přímky (3.6).

$$k_i = 0.224 * d_i - 0.0333, \tag{3.6}$$

kde k_i je distanční koeficient a d_i vzdálenost pro i-tý bod. Koeficienty byly odvozeny podle naměřených dat. S aktuální nastavenou rychlostí měření bylo změřeno, že ve vzdálenosti jednoho metru je průměrná rozteč mezi body 22,3 mm.



Obrázek 3.10: Odvození distančního koeficientu.

Hledané rysy se neukládají také v případě, že délka vektorů na bod překračuje $1,5 * k_i$.

V případě hledání hran se používá modifikovaný algoritmus *Split and Merge*. V obyčejném algoritmu se dělicí body hledají podle největší kolmé vzdálenosti. Modifikace spočívá v tom, že se nehledá největší kolmá vzdálenost, ale přímka se rozdělí v případě, že překročí požadovanou přesnost. Aby množina bodů byla považována za hranu musí splnit několik podmínek. První podmínka je velikost aproximační chyby přímky S_E (3.7) a její hodnota musí být menší než 3. Další podmínka je velikost r^2 (koeficient of determination) (3.8), ta musí být alespoň 98% a udává přesnost aproximace přímky danými body.

$$S_E = \sum_{i=1}^{n} (y_i - (kx_i + q))^2. \tag{3.7}$$

$$r^2 = 1 - \frac{S_E}{S_{Ey}}, \tag{3.8}$$

kde S_{Ey} je chyba ve směru y a vypočte se jako $S_{Ey} = \sum_{i=1}^{n}(y_i - \bar{y})^2$ a $\bar{y} = \frac{1}{n}\sum_{i=1}^{n}(y_i)$ je střední hodnota.

Důležité je také kontrolovat minimální délku hrany, aby nevznikaly hrany o délce tří bodů atp., proto je přidána podmínka, že minimální délka přímky je 6 bodů. Při hledání další možné hrany se také kontroluje, vzdálenost sousedících bodů, aby se jednalo o souvislou hranu. Vzdálenost sousedících bodů, může být maximálně $2 * k$, kde $k = (k_i + k_{i-1})/2$ je aritmetický průměr distančních koeficientů sousedicích bodů.

Pokud je nalezena hrana, aplikuje se na ní ještě funkce, která se snaží snížit její chybu, tím že zkouší odstranit bod ze začátku nebo z konce. Tato úprava se aplikuje pouze v případě, že se hodnota chyby, přepočítané na bod, sníží.

Jako důležité objekty se ukládají také, průsečíky všech nalezených přímek. Myšlenka je taková, že pokud je nalezená hrana dostatečně přesná, tak průsečíky dvou hran při dalším měření budou na stejném místě, nebo v malém okolí tohoto bodu.

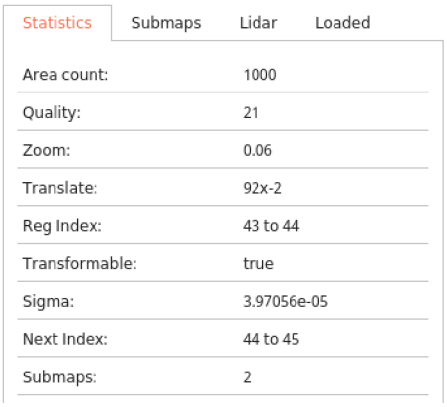
3.5.2. Registrace mračen bodů

Vstupem do algoritmu jsou dva mračna bodů. V průběhu se jeden soubor bodů snaží zarovnat k druhému, tím že se zkouší vypočítané transformace. Na začátku se oba mračna bodů převedou na normované (bezrozměrné) veličiny.

Souvislý posun bodů je iterativní algoritmus, kde ukončující podmínky jsou nastaveny následovně. První podmínka je maximální počet iterací nastavený na 100, další je velikost rozptylu σ^2, který musí být menší než $4e^{-5}$. Předposlední je počet iterací kdy se má ukončit algoritmus pokud se σ^2 nezmění. Tato hodnota je nastavena na 10. Poslední podmínka je později přidána, kdy se odvodilo, že pokud algoritmus dostatečně nezkonverguje do 30. iterace (tzn. $\sigma^2 < 1e^{-4}$), tak je zarovnávaní ukončeno.

Prvním krokem každé iterace je výpočet matice rozdělení pravděpodobnost $\mathbf{P}$ pomocí FGT (rychlá Gaussova transformace). Tato transformace nevypočítá matici $\mathbf{P}$ jako takovou, ale matice $\mathbf{P1}, \mathbf{Pt1}$ *a* $\mathbf{PX}$, které jsou potřebné k dalšímu výpočtu. Dále se pokračuje už stejně jak v modifikaci pro tuhé transformace, kdy v každé iteraci se vypočte rotační matice $\mathbf{R}$, translační vektor $\mathbf{t}$ a měřítko s. Z těchto hodnot se určí rozptyl σ^2, který se ověřuje v ukončujících podmínkách. Pokud žádná podmínka není splněna, výpočet pokračuje další iterací.

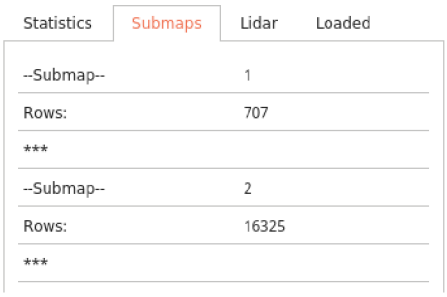
Registrace probíhá ve třech vláknech, kdy první dva pouze počítají hodnoty transformací pro všechny oblasti (area_t) a třetí vlákno prochází výsledky. Pokud je zarovnání možné, třetí vlákno danou oblast přidá do mapy. Výsledky jednotlivých transformací jsou průběžně zobrazovány v pravém sloupci aplikace (obr. 3.11), kde aktuální transformace je na řádku 5 (Reg. Index). Další řádek představuje úspěšnost transformace spolu s výsledným rozptylem σ^2. Na předposledním řádku jsou indexy přístí transformace a na konci je aktuální počet submap.



Obrázek 3.11: Registrace mračen bodů, statistiky.

Pokud třetí vlákno narazí na oblasti, které se nepodařilo spojit, spustí korekční část, kdy se snaží danou oblast spojit s nějakou předchozí. Takto to zkouší až na 3 předešlé oblasti a pokud se to nepovede, vytvoří se nová submapa. V pravém menu aplikace pod záložkou *Submaps* (obr. 3.12) je seznam všech submap, kde po kliku se vykreslí.

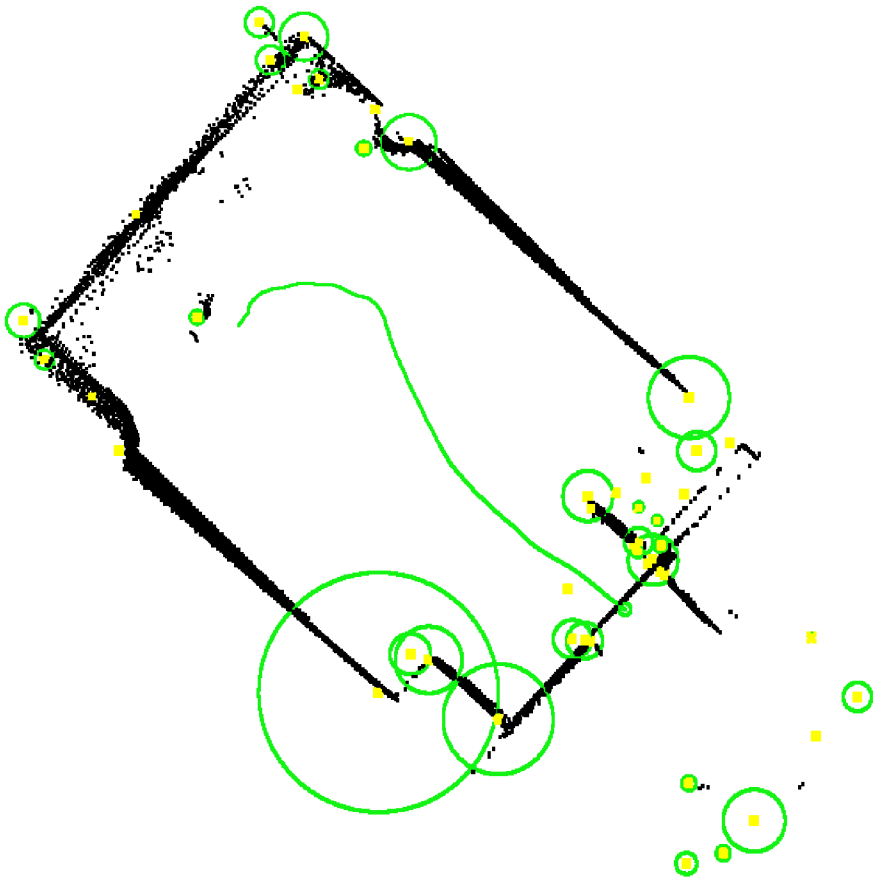
Po každém spojení oblastí se provádí hledání stejných důležitých rysů v okolí dosud nalezených. Pokud se najde takovýto objekt, zvyší se jeho důvěryhodnost o 1 a zpřesní



Obrázek 3.12: Seznam uložených submap.

se jeho pozice. Zpřesnění pozice se provádí v tom smyslu, že větší váhu má nová pozice. Na obrázku 3.13 je zobrazena věrohodnost objektů pomocí zelených kružnic, kdy čím větší průměr, tím vyšší věrohodnost. Na tomtéž obrázku je vykreslena také cesta robota, kterou urazil (zelená křivka). Robota symbolizuje zelené kolečko na konci.

Zajímavé na obrázku 3.13 jsou důležité objekty vykresleny mimo, tam kde nejsou téměř žádné body. Tento jev je zapříčiněn tím, že v aplikaci se používa dvě mapy. Jedna je pouze na vykreslování a je zbavena nepřesných bodů. Nepřesné body jsou ty, které leží dál od LIDARu. Bylo zjištěno, že body vzdálenější než 1500 mm bývají už nepřesné. Ke zpracování se ale používá druhá mapa, v které jsou body všechny a proto některé objekty leží mimo zobrazené body.



Obrázek 3.13: Důvěryhodnost objektů vizualizována jako zelené kružnice.

3.6. Připojení k serveru

LIDAR je obsluhován stavovým automatem naprogramovaném v jazyce Python a běží na Raspberry Pi (serverová část). Kvůli vývoji se dá server spustit i na lokální síti, kde IP adresa je "127.0.0.1". V aplikaci se používá několik stavů, v kterých se zrovna nachází. Výchozí stav je nepřipojená aplikace k serveru. K připojení se v záložce *Nastavení* (Settings) nachází tlačítko *Lidar*. Po kliknutí se obsahové části zobrazí nastavení k použití UDP protokolu. UDP je nepotvrzovaný protokol používající datagramy pro přenost dat. Hlavní výhodou nepotvrzování je zvyšená rychlost a menší zátěž sítě (vyšší propustnost). K základnímu nastavení je potřeba tři údaje, první je IP adresa serveru, dále zdrojový port (ZP) a poslední cílový port (CP, obr. 3.14). Porty jsou dva z toho důvodu, že jeden se používá k odesilání a druhý k přijímání. Před odesláním paketu se ještě přidává údaj o délce dat (N) a kontrolní součet (S).

ZP	CP	N	S	DATA

Obrázek 3.14: Paket při použití UDP protokolu.

K navázání spojení se serverem se posílá přesně definovaný datagram o délce 7 bajtů a obsahuje následující data (obr. 3.15). Na začátku je jeden bajt s adresou zařízení, tzn. pro adresování serveru je to 255. Dále následuje bajt s příkazem, kde pro inicializaci připojení je nastaven na 0. Na konec datagram obsahuje zprávu, kde při inicializaci je dobrým zvykem posílat zprávu "HELLO".

255	0	H	E	L	L	O

Obrázek 3.15: Datagram k navázání spojení se serverem.

Pokud na serveru není nikdo připojený a tedy může otevírat nová připojení, odpoví na předem zadaný port. V odpovědi odešle seznam právě dostupných zařízení (obr. 3.16). Server je naprogramovaný univerzálně, a proto může obsluhovat více věcí, jako například LIDAR, GPIO (piny na Raspberry Pi) a další senzory. Přijatý datagram má opět definovanou strukturu, kde první bajt je 255, druhý je 1 (příznak pro seznam zařízení) a následují data. Pro získání seznamu je potřeba data rozdělit. Dané moduly (zařízení) jsou odděleny pomocí "##"a každý modul má dvě položky oddělené pomocí "::". Modul obsahuje své ID a jméno na serveru, ID je dále potřeba pro práci s tímto zařízením.

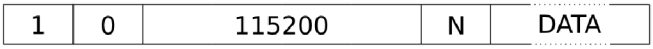
255	1	DATA

Obrázek 3.16: Odpověď od serveru při navázání spojení.

Po obdržení tohoto datagramu, server nepřijímá žádné další inicialzační požadavky. Pro udržení připojení je potřeba odesílat tzv. "heartbeat", to v praxi vypadá tak, že se každou sekundu odešle na server zpráva, že je klient pořád aktivní. Datagram obsahuje pouze

dva bajty, kde první je 255 a druhý 2 (příznak pro heartbeat). Pokud server neobdrží tuto zprávu po dobu 5 sekund, je klient odpojen.

Dále už bude popsána pouze komunikace s serverem k obsluhování LIDARu. Pro práci se skenerem je na serveru stavový automat, který je ve výchozím nepřipojeném stavu. V aplikaci je potřeba zadat cestu k skeneru, v případě linuxu je přednastavená hodnota "/dev/ttyUSB0". Po kliku na tlačítko *Connect to Lidar*, je na server odeslán datagram s velikostí přenosové rychlosti a umístění (obr. 3.17). Datagram začíná prvním bajtem s adresou zařízení (1 pro LIDAR), další je bajt s příkazem (0 - navázání spojení), dále čtyři bajty s přenosovu rychlostí. Přenosová rychlost je v aplikaci pevně nastavena na 115200 Bd. Následující bajt obsahuje údaj o délce dat, které obsahují cestu k LIDARu.



Obrázek 3.17: Datagram pro přechod do připojeného stavu.

Po připojení k skeneru, server odešle zprávu s detaily o aktuálním stavu. Délka datagramu jsou 3 bajty, kde první je 1 pro LIDAR, druhý 4 pro příznak, že se jedná o zprávu o stavu. Poslední bajt je stav a ten může nabývat těchto hodnot:

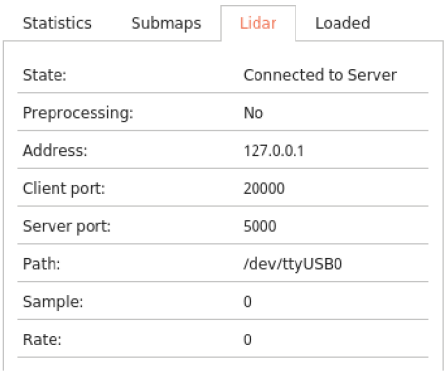
- 0 - Čekání, nepřipojeno,
- 1 - Připojeno,
- 2 - Měření zapnuto,
- 3 - Odesílání dat,
- 4 - Chybový stav.

V aplikaci se dále zobrazí ovládání skeneru, kde je potřeba zadat požadovaný počet vzorků a interval, v kterém se budou měření odesílat. V tomto okamžiku je aplikace připojena k LIDARu. Pro začátek skenování a příjem naměřených dat, stačí kliknout na tlačítko *Start*. Aktuální stav je zobrazen ve statistikách (pravé menu záložka Lidar, obr. 3.18), obsahuje informace o používaných portech, IP adresu serveru, cestu k skeneru a údaje o měření.

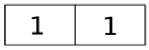
V připojeném stavu pro začátek skenování stačí odeslat na server následující datagram (obr. 3.19). První bajt opět signalizuje komunikaci s LIDARem a druhý bajt je pro požadavek začít měřit (v tomto stavu ještě neodesílá data do aplikace). Pokud je potřeba měření zastavit, tak se místo 1 na druhém bajtu odesílá 2.

Server začne odesílat data až pokud jsou požadovány (obr. 3.20). Datagram je potom v tomto tvaru, první bajt je 1, druhý 3 (odesilání dat), následují čtyři bajty s požadovaným počtem měřených vzorků (VZORKY) a poslední čtyři bajty symbolizují požadovaný interval (INT).

Po odeslání tohoto datagramu, server začne posílat data v nastaveném intervalu. Měření má také daný tvar (obr. 3.21), kde první dva bajty jsou stejné. Další čtyři bajty obsahují údaj o počtu vzorků (k určení délky dat). Každý jeden vzorek má velikost 5 bajtů



Obrázek 3.18: Dostupné statistiky informující o aktuálním stavu připojení.



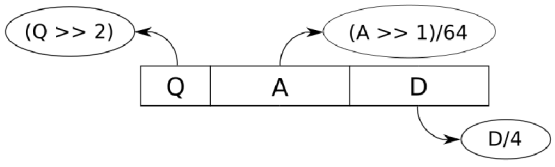
Obrázek 3.19: Datagram pro začátek měření.

Obrázek 3.20: Datagram s požadavkem posílat měření ze serveru.

a v těch je uložena hodnota úhlu, kvality a vzdálenosti. První bajt je hodnota kvality, další dva bajty úhel a nakonec dva bajty se vzdáleností.

Obrázek 3.21: Datagram s aktuálním měřením.

LIDAR odesílá všechny hodnoty jako celá čísla a ty je potřeba přepočítat (obr. 3.22). K dekódování kvality je potřeba na tento bajt provést bitový posun doprava o 2 bity. Bajty s úhlem je potřeba bitově posunout taktéž doprava, ale pouze o jeden bit a následně vydělit hodnotou 64. U vzdálenosti se pouze hodnota z daných bajtů vydělí čtyřmi.



Obrázek 3.22: Přepočet hodnot na kvalitu, úhel a vzdálenost.

3.7. Zpracování dat přímo z LIDARu

Pokud je LIDAR v připojeném stavu, záložka *Real Time Render* je aktivní a je možné začít zpracovávat přijatá data. Jsou dostupné pouze tlačítka pro start, stop a přepínání mezi zobrazením mapy a aktuálního měření. Po kliku na *Start* se nastaví hodnoty tak, že se začne aplikovat registrace a hledání důležitých objektů na přijatá data. Opět jsou spuštěny tři vlákna, kde třetí skládá výslednou mapu.

Zpracování v reálném čase potřebuje velký výpočetní výkon, z toho důvodu není vždy zaručeno, že algoritmus bude počítat v reálném čase. Na tuto skutečnost má také vliv nastevený interval odesílání měření, kdy pro slabší PC se může nastavit 500 ms až 1000 ms. Zvyšování intervalu, ale může vést i k nestabilitě algoritmu, protože to také záleží na rychlosti pohybu skeneru.

3.8. Požadavky k spuštění aplikace

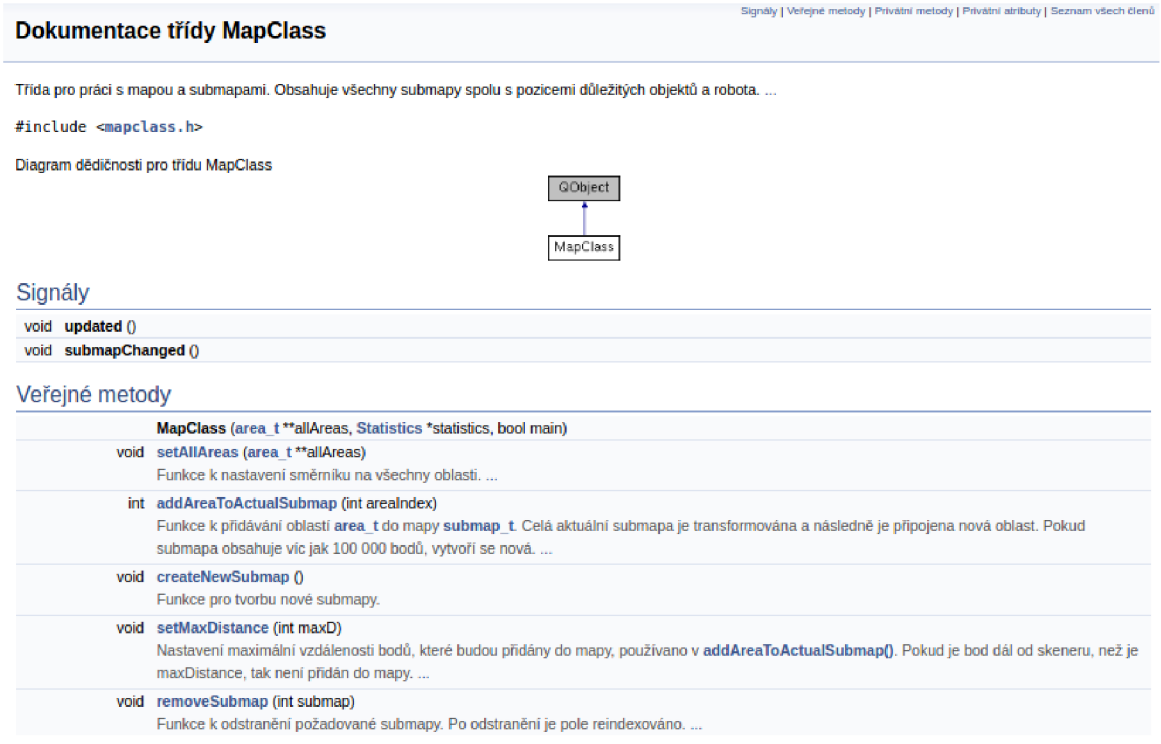
Nejdůležitější požadavek je *QT Creator*, neboť aplikace není dodáváná i s instalačním softwarem. Z toho důvodu je potřeba aplikaci spouštět ze zmiňované aplikace. Při použití operačního systému Windows je doporučené doinstalovat si kompilátor MinGW (dostupné při instalaci QT Creatoru).

K použití daných algoritmů, je potřeba také doinstalovat některé knihovny. První knihovna je *Armadillo*, která obsahuje mnoho matematických funkcí a proměnných jako matice, vektory a další. Také je potřeba knihovnu pro použití rychlé Gaussovy transformace *FGT*, která je dostupná na GitHub [24] (je potřeba verzi 0.2). Nejlepší postup jak přidat *FGT* k projektu, je zdrojové soubory knihovny nakopírovat k aplikaci a v *QT Creatoru* pouze přidat. Nevýhodou je, že obě knihovny nemají instalační prostředí pro Windows a tedy je potřeba je zkompilovat ručně (nejlépe pomocí *Cmake* a *Visual Studia*). V operačním systému Windows lze nahradit nebo rozšířit *Armadillo* pomocí knihoven *BLAS* a *LAPACK*, které jsou dostupné předkompilované [25].

3.9. Dokumentace k programu

K aplikaci je dostupná i dokumentace (obr.3.24), která obsahuje popis všech používaných tříd a metod. Dokumentace je ve formátu HTML a byla vytvořena pomocí Doxygen [26].

Obrázek 3.23: Seznam tříd v dokumentaci k aplikaci.



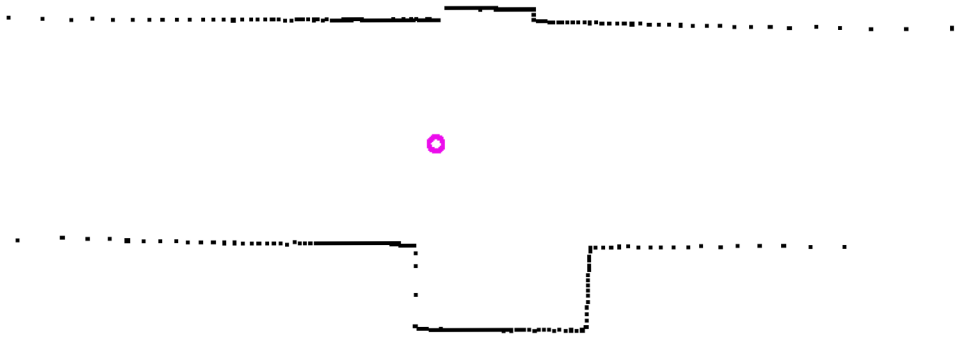
Obrázek 3.24: Dokumentace k třídě MapClass.

4. Demonstrace řešení v modelových prostředích, vyhodnocení

V této kapitole jsou demonstrována výsledná řešení a jejich nedostatky. K měření byl použit RPLIDAR od firmy RoboPeak a data byla zpracována pomocí aplikace vytvořené k této diplomové práci. Při měření nebyly použity žádné další senzory. Měření byla provedena v modelovém prostředí objektu FSI A3 7. patro.

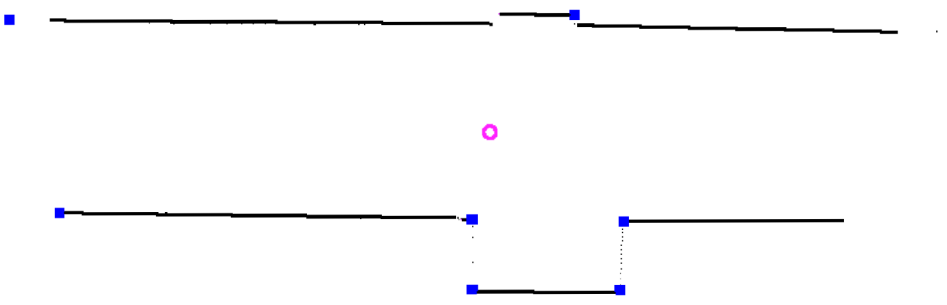
4.1. Hledání důležitých objektů

Jedno mračno naměřených bodů má průměrně 280 bodů (v používaném nastavení), s ohledem na kvalitu příchozích pulzů. Při vykreslení pouze jednoho měření (obr. 4.1) jsou body vyznačeny červeně a pozice skeneru (robota) zeleným kolečkem.



Obrázek 4.1: Vykreslení jednoho měření (jedna otáčka skeneru).

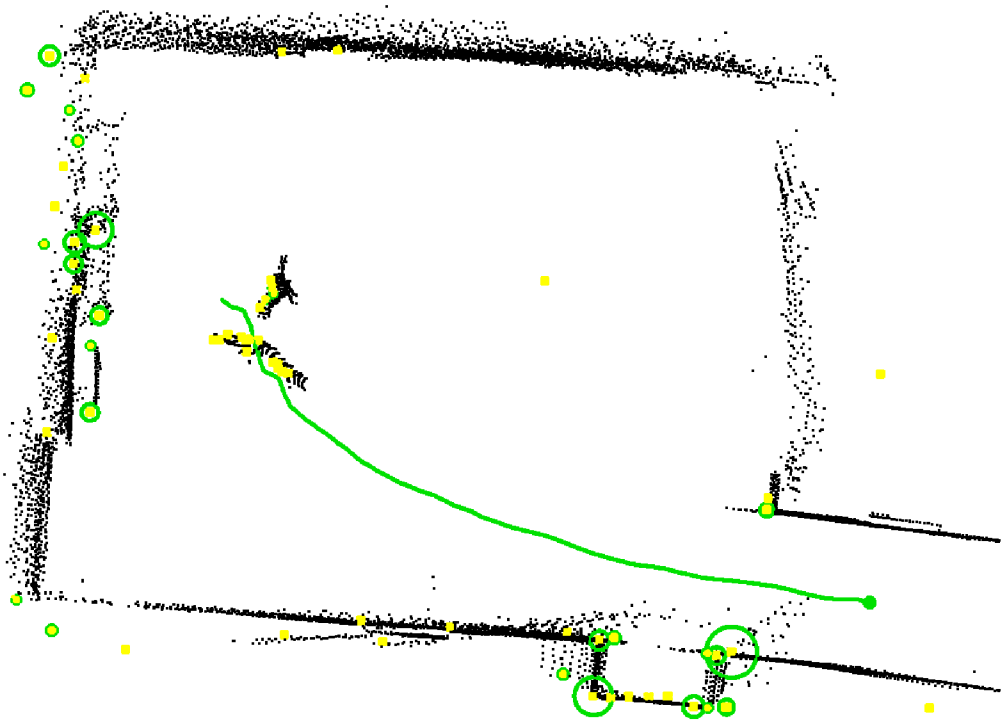
Po zapnutí hledání duležitých objektů, se na aktuálním měření spustí algoritmus. Důležité objekty typu bodů jsou vyznačeny většími body a hrany jsou vyznačeny přímkami.



Obrázek 4.2: Hledání důležitých objektů v jednom měření.

4.2. Registrace a vytváření mapy prostředí

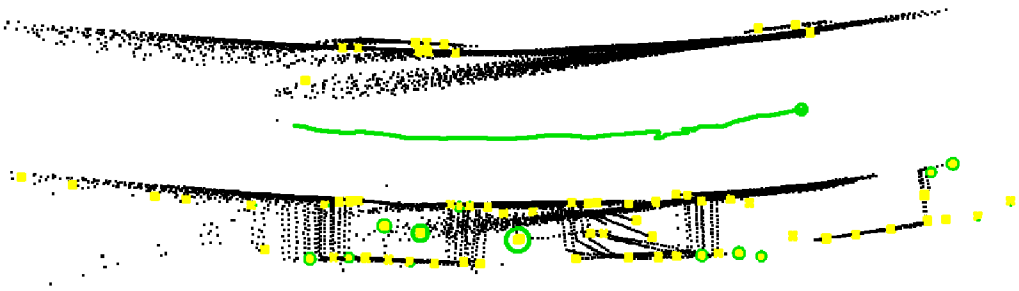
Při zapnutém vytváření mapy se celé mračno bodů automaticky překresluje při každé přidané oblasti. V případě měření chodby (obr. 4.3) se jedná o mapu, která má přes 18800 bodů. Na této mapě je vidět případ, kdy se začíná téměř uprostřed, a proto jsou stěny na začátku rozostřené. Jak měření pokračuje skener se pohybuje dolů ke stěně a do chodby kde je vidět jak se hrany vyhlazují. Při tomto pohybu se také upravuje pozice důležitých bodů.



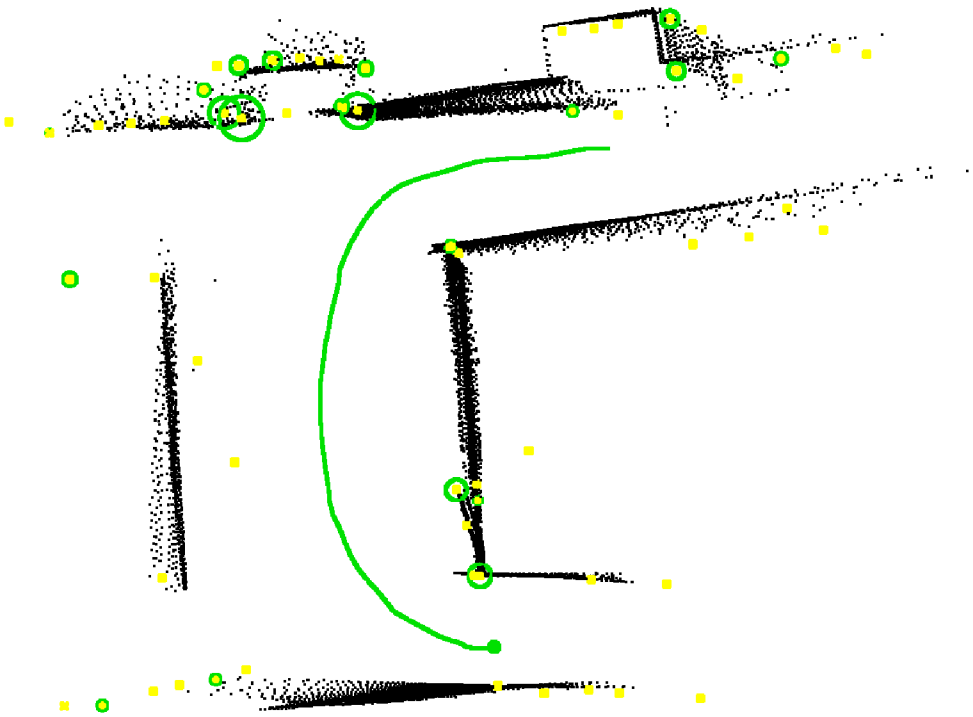
Obrázek 4.3: Měření začátku chodby (modelové prostředí FSI A3 7. patro).

Na dalším obrázku (obr. 4.4) se nachází pokračování chodby. V tomto případě se jedná, ale o špatné spojení. Ve skutečnosti je chodba delší, ale z toho důvodu, že se používá pouze jeden senzor, dochází ke zkreslování. Algoritmus nemá v některých případech žádné záchytné body (pouze rovnou stěnu) a tak výsledkem je nulová translační složka.

Tuto chybu lze odstranit přidáním dalšího senzoru, který by měl větší dosah, nebo doplnit robota o měření pohybu přímo na kolech. Chodba dále pokračuje (obr. 4.5), začátek submapy je stále deformován dlouhou částí, ale následně se mapa vykresluje správně. Na první pohled se může zdát, že důležitých objektů je tam hodně, ale většina z nich má nízkou důvěryhodnost. Pokud by se pracovalo pouze s objekty s vyšší důvěryhodností, zůstalo by jich tam pouze asi dvacet.

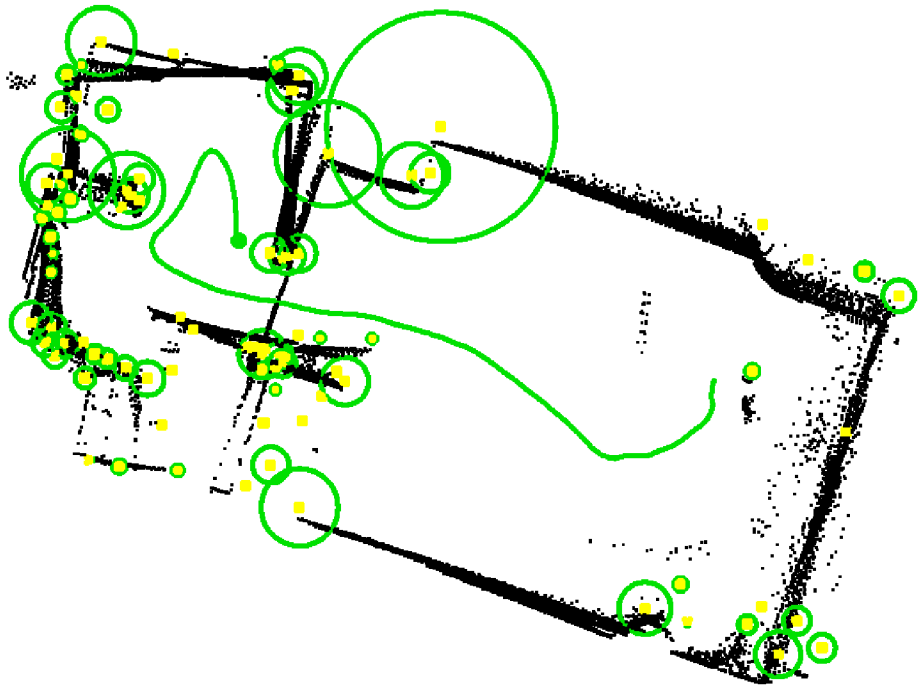


Obrázek 4.4: Pokračování měřené chodby (modelové prostředí FSI A3 7. patro).



Obrázek 4.5: Měření konce chodby (modelové prostředí FSI A3 7. patro).

Další je měřená kancelář (obr. 4.6), na tomto měření je demostrováno co se stane, pokud se přiřadí jedno měření, které mělo špatně spočítanou transformaci. První část je bez problémů, problém nastal až při průchodu dveřmi, kdy jedno měření způsobilo špatné natočení mapy. Z toho důvodu je hodně důležitých bodů na jednom místě, které by při správném spojení zvyšovaly pouze důvěryhodnost, tak dochází k duplikování.



Obrázek 4.6: Měření kanceláře (modelové prostředí FSI A3/720).

Tyto chyby mohou nastat kdykoliv a špatně se detekují, protože míra správného spojení se reprezentuje velikostí rozptylu σ^2. Při měřeních bylo zjištěno, že pokud je rozptyl větší než $4e^{-5}$ tak se registrace mračen bodů nepovedla a algoritmus uvízl v lokálním extrému. Nicméně této chybě lze do značné míry předejít a to kontrolou transformace podle důležitých objektů.

5. Závěr

Cílem práce bylo popsat možnosti využití SLAMu spolu s algoritmy, které se k tomuto účelu používají. Následně pomocí zvoleného senzoru LIDAR naměřit data a ty zpracovat pomocí vlastní aplikace. V aplikaci bylo potřeba mračna bodů správně zpracovat a vytvořit mapu měřeného prostředí.

V první kapitole jsou představeny používané algoritmy pro extrahování důležitých objektů z mračen bodů. Algoritmy se dělí do skupin podle použití. První skupina jsou metody k hledání důležitých bodů neboli rohů. Druhou skupinou je hledání hran (přímek), ty ve skutečnosti představují stěny a předměty podobného charakteru. Kapitola dále vysvětluje několik metod, které se používají pro hledání transformací na dva mračna bodů. Transformace se hledají z toho důvodu, že je potřeba určit jak se změnila pozice robota mezi měřeními. Algoritmy pro zarovnávání bodů jsou vypsány i s jejich výhodami a nevýhodami. K hledání rohů byl zvolen algoritmus *Sliding window*, k hledání přímek potom algoritmus *Split and merge* a pro registraci mračen bodů *Zarovnávání souvislým posunem bodů*.

V druhé kapitole je popsáno několik různých LIDARů spolu s používaným senzorem k této diplomové práci. Také obsahuje popis komunikačního protokolu, který daný LIDAR používá.

Následující třetí kapitola se věnuje programové implementaci, která byla provedena v QT frameworku v jazyce C++. Jsou zde představeny hlavní používané struktury a funkcionalita programu. Aplikace je koncipována tak, že má dva vstupy pro data a to buď lze nahrát předem naměřená data nebo se připojí k serveru, ke kterému je připojený LIDAR. Byl implementován UDP protokol pro komunikaci se serverem a příjem naměřených dat. Také byly implementovány zvolené algoritmy na extrahování důležitých objektů a zarovnávání mračen bodů. V aplikaci jsou průběžne zobrazovány statistiky a je možnost zobrazit také výslednou sestavenou mapu. Z aplikace je možné také ukládat a nahrávat mapy pro pozdější vykreslení a využití.

Čtvrtá kapitola se věnuje demonstraci aplikace, kde jsou naměřené různé prostory. Na obrázcích jsou správně zarovnané oblasti, ale také oblasti, které byly spojeny s chybou. Tyto chyby vznikají hlavně z důvodu absence dalších senzorů (s větším dosahem, měření pohybu).

V naváznosti na možnost rozšíření a vylepšení aplikace jsou zde představeny možné vylepšení. Vylepšení ze strany hardwaru, jak už bylo zmiňováno, je přidání dalších senzorů ke kontrole výsledných transformací (k zlepšení indoor navigace). Ze strany softwaru je možnost implementovat ještě zarovnávání na základě důležitých objektů (s vyšší důvěryhodností), také přidat informace o rozměrech mapy a používat nahrané mapy ke srovnání s měřením. Na tento typ zarovnávání je potřeba použít jiny algoritmus, protože většina zde představených algoritmů je při použití na tak nízký počet bodů nestabilní. Také by bylo vhodné aplikaci doplnit o detekci pohyblivých objektů, protože teď úspěšnost zarovnání silně závísí na přítomnosti pohyblivých objektů a na jejich vzdálenosti od senzoru.

Závěrem je možné konstatovat, že všechny body zadání diplomové práce byly naplněny. Byla vytvořena aplikace, která zarovnává a zpracovává naměřená mračna bodů. Z těchto naměřených dat vytváří výslednou mapu prostředí.

Literatura

[1] RIISGAARD, Søren a BLAS, Morten Rufus. *SLAM for Dummies* [online]. [cit. 2016-04-13]. Dostupné z: http://ocw.mit.edu/courses/aeronautics-and--astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf

[2] SZELISKI, Richard. *Computer Vision: Algorithms and Applications* [online]. 2010 [cit. 2016-04-13]. Dostupné z: http://szeliski.org/Book/

[3] NAMOSHE, Molaletsa; MATSEBE, Oduetse a TLALE, Nkgatho. *Corner Feature Extraction: Techniques for Landmark Based Navigation Systems, Sensor Fusion and its Applications*. INTECH, 2010. 488 p. ISBN 978-953-307-101-5. Dostupné z: http://www.intechopen.com/books/sensor-fusion-and-its- applications/corner-feature-extraction-techniques-for-landmark-based-navigation-systems

[4] DERPANIS, Konstantinos G. *Overview of the RANSAC Algorithm* [online]. 2010 [cit. 2016-04-14]. Dostupné z: http://www.cse.yorku.ca/k̃osta/CompVis_Notes/ransac.pdf

[5] THURLEY, Matthew. *The Hough Transform* [online]. [cit. 2016-04-16]. Dostupné z: http://www.ltu.se/cms_fs/1.36192!/e0005e_lecture05_hough_transform.dvi.pdf

[6] Hashovací tabulka. *ITnetwork.cz* [online]. 2013 [cit. 2016-04-18]. Dostupné z: http://www.itnetwork.cz/algoritmy/vyhledavani/algoritmus-vyhledavani-hashovaci-tabulka/

[7] *Kd-stromy* [online]. [cit. 2016-04-18]. Dostupné z: http://phoenix.inf.upol.cz/k̃onecnja/vyuka/2013/ALS1files/new/05b.pdf

[8] *2D Transformation* [online]. [cit. 2016-04-18]. Dostupné z: http://www.tutorialspoint.com/computer_graphics/2d_transformation.htm

[9] FUSSEL, Don. *Affine Transformations* [online]. 2010 [cit. 2016-04-18]. Dostupné z: https://www.cs.utexas.edu/f̃ussell/courses/cs384g/lectures/lecture07-Affine.pdf

[10] BURGARD, Wolfram. *Iterative Closest Point Algorithm* [online]. [cit. 2016-04-18]. Dostupné z: http://ais.informatik.uni-freiburg.de/teaching/ss11/robotics/slides/17--icp.pdf

[11] JANČÍK, S.; MATOUŠEK, R.; DVOŘÁK, J.; ABBADI, A. *The ICP for Fragment Identification*. In 18th International Conference of Soft Computing, MENDEL 2012. Mendel Journal series. 2011. Brno: VUT, 2012. s. 588-593. ISBN: 978-80-214-4540-6. ISSN: 1803- 3814.

[12] HEIKKILA, J. *A statistical method for object alignment under affine transformation*. Proceedings 12th International Conference on Image Analysis and Processing. 2003. IEEE. s. 360-365. ISBN: 0-7695-1948-2.

[13] HO, Jeffrey; YANG, Ming-Hsuan; RANGARAJAN, Anand; VEMURI, Baba. *A New Affine Registration Algorithm for Matching 2D Point Sets* [online]. 2007 [cit. 2016-04-22]. Dostupné z: https://www.cise.ufl.edu/ãnand/pdf/WACV_Final.pdf

[14] MYRONENKO, Andriy a SONG, Xubo. *Point Set Registration: Coherent Point Drift* [online]. 2009 [cit. 2016-04-24]. Dostupné z: https://arxiv.org/pdf/0905.2635.pdf

[15] JIAN, Bing. *Robust Point Set Registration Using Gaussian Mixture Models*. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2010. IEEE. s. 1633-1645. ISBN: 0162-8828.

[16] WANG, Chieh-Chih. *Simultaneous Localization, Mapping and Moving Object Tracking* [online]. 2007 [cit. 2016-04-23]. Dostupné z: https://www.ri.cmu.edu/pub_files/pub4/wang_chieh_chih_2007_1/wang_chieh_chih_2007_1.pdf

[17] LIDAR-Lite 2 Laser Rangefinder. *RobotShop* [online]. [cit. 2016-04-25]. Dostupné z: http://www.robotshop.com/en/lidar-lite-2-laser-rangefinder-pulsedlight.html

[18] 2D laser scanners. *SICK* [online]. [cit. 2016-04-25]. Dostupné z: https://www.sick.com/de/en/product-portfolio/detection-and-ranging-solutions/2d-laser-scanners/c/g91900

[19] Hokuyo UXM-30LN-PW Scanning Laser Rangefinder. *RobotShop* [online]. [cit. 2016-04-25]. Dostupné z: http://www.robotshop.com/en/hokuyo-uxm-30ln-pw-scanning-laser-rangefinder.html

[20] Sweep LIDAR. *Scanse* [online]. [cit. 2016-04-25]. Dostupné z: http://scanse.io/

[21] 360° LIDAR Development Kit. *SLAMTEC* [online]. Dostupné z: http://www.slamtec.com/en/Lidar/A1

[22] HDL-64E. *Velodyne LiDAR* [online]. [cit. 2016-04-25]. Dostupné z: http://velodynelidar.com/hdl-64e.html

[23] *QT*. [online]. [cit. 2016-05-01]. Dostupné z: http://www.qt.io/

[24] Fast Gauss transforms. *GitHub* [online]. [cit. 2016-05-11]. Dostupné z: https://github.com/gadomski/fgt

[25] *LAPACK for Windows* [online]. [cit. 2016-05-11]. Dostupné z: https://icl.cs.utk.edu/lapack-for-windows/lapack/

[26] *Doxygen* [online]. [cit. 2016-05-13]. Dostupné z: http://www.stack.nl/ dimitri/doxygen/