VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PRAVDĚPODOBNOSTNÍ LOKALIZACE MOBILNÍHO ROBOTA

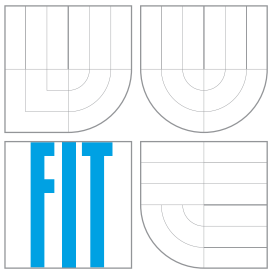
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ KOPEČNÝ

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PRAVDĚPODOBNOSTNÍ LOKALIZACE MOBILNÍHO ROBOTA

PROBABILISTIC LOCALIZATION OF MOBILE ROBOT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — TOMÁŠ KOPEČNÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAROSLAV ROZMAN,
SUPERVISOR

BRNO 2012

Abstrakt

Řízení pohybu, navigace a orientace robota v prostoru jsou problémy spjaté s vývojem mnoha robotů. Tato práce se zabývá především lokalizací ve smyslu orientace robota v prostoru, nicméně ani ostatní aspekty nejsou opomenuty. Cílem práce je konkrétně lokalizační algoritmus Monte Carlo, jehož přiblížením a aplikací na skutečného robota se zde budeme zabývat.

Abstract

Motion control, navigation and sense of orientation of a mobile robot are tied to development of every mobile robot. This work concerns mainly about robot localization in the sense of orientation in space. Nevertheless, nor other aspects of mobile robotics are being omitted. Therefore, the goal of this work is to get familiar with Monte Carlo localization algorithm and apply this algorithm on a real robot.

Klíčová slova

Robot, lokalizace robota, pravděpodobnost, Monte Carlo, simulace, hledání cesty, řízení pohybu, odometrie, sériové rozhraní, sériová komunikace, FITkit.

Keywords

Robot, localization of robot, probability, Monte Carlo, simulation, pathfinding, motion control, odometry, serial interface, serial communication, FITkit.

Citace

Tomáš Kopečný: Pravděpodobnostní lokalizace mobilního robota, bakalářská práce, Brno, FIT VUT v Brně, 2012

Pravděpodobnostní lokalizace mobilního robota

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Rozmana

.....
Tomáš Kopečný
16. května 2012

Poděkování

Chtěl bych poděkovat vedoucímu panu Ing. Jaroslavu Rozmanovi za jeho dohled při řešení úlohy. A zvláště bych chtěl poděkovat panu Ing. Tomáši Novotnému, za jeho pomoc při zprovoznění robota.

Obsah

Kapitola 1

Úvod

V současné době je téma robotiky velice aktuální. S řízením robotů je však spjato mnoho problému, které je potřeba efektivně řešit. Jedním z nich je problém lokalizace, neboli schopnosti robota se samostatně orientovat a pohybovat v prostoru. Jako vhodné se ukázují algortimy využívající pravděpodobnost pro reprezentaci odhadu skutečné pozice robota [11].

V kapitole 2 se s pravděpodobnostní lokalizací blíže seznámíme a stručně si přiblížíme základní algoritmy tuto problematiku řešící.

Hlavním cílem této práce je seznámení se s lokalizačním algoritmem Monte Carlo, založeném na pravděpodobnostní reprezentaci odhadu pozice robota ve spojení s generátory náhodných čísel. Podrobně o tomto algoritmu se dočtete v kapitole 3.

Ve čtvrté kapitole bude navržena aplikace, využívající tento algoritmus pro lokalizaci skutečného robota vyvíjeného na UITS, FIT VUT v Brně [7][9]. V souvislosti s autonomní orientací robota, je potřeba řešit také otázku jeho řízení a hledání cesty. Přesto, že toto téma není cílem této práce, při lokalizaci skutečného robota se tomu nelze vyhnout, proto i tímto problémem se budeme zabývat v kapitole 4.

V kapitole páté uvedeme jaké prostředky byly použity k implementaci navržené aplikace. Je zde také popsána struktura zdrojových kódů aplikace.

Kapitola šestá se věnuje ovládání výsledné aplikace.

Poslední kapitola je shrnutím a zhodnocením dosažených výsledků. Také je zde zmíněno několik návrhů budoucí práce související se školním robotem.

Kapitola 2

Seznámení s problematikou lokalizace mobilního robota

V této kapitole se čtenář seznámí s tím, co to je lokalizace mobilního robota. A dále budou stručně představeny nejznámější algoritmy řešící tuto problematiku.

2.1 Lokalizace mobilního robota

Lokalizací mobilního robota se rozumí autonomní orientace robota v prostoru. To znamená schopnost robota se samostatně a cíleně pohybovat v prostředí, ve kterém se nachází a udržovat aktuální informaci o své poloze.

Existují dva typy lokalizace. Liší se tím, jestli robot má nebo nemá k dispozici popis (mapu) prostředí, ve kterém se nachází.

2.1.1 Požadavky na robota

Aby mohl robot správně provádět lokalizaci musí mít prostředky jak se pohybovat. Avšak samotná schopnost pohybu pro správnou funkci algoritmů lokalizace nestačí. Důležitá je i zpětná vazba od těchto pohybových součástek ve formě informací o relativní změně polohy. Proces získávání těchto informací se nazývá *odometrie*.

Dalším předpokladem pro úspěšné řešení problému lokalizace je, že je robot vybaven senzory, které mu umožňují získávat informace o jeho okolí (ať už jde o informace o vzdálenostech, teplotě, či jiných fyzikálních veličinách)

2.1.2 Pravděpodobnostní reprezentace odhadu skutečné pozice

Reprezentace pozice robota pomocí pravděpodobnosti, je velice žádoucí z toho hlediska, že umožňuje vyjádřit nejistotu v tom zda odhadnutá pozice odpovídá skutečnosti. Je navíc vhodné být schopen reprezentovat naprosto libovolné rozložení pravděpodobnosti, což některé algoritmy neumožňují.

2.1.3 Vlastnosti lokalizačních algoritmů

Lokalizační algoritmy musí splňovat několik požadavků. Patří sem schopnost pracovat v reálném čase, schopnost reprezentovat aktuální pozici robota prostřednictvím pravděpo-

dobnosti, schopnost reagovat na dynamické (časově proměnné) prostředí.

Schopnost práce v reálném čase vede k potřebě hledat algoritmy velice efektivní, protože samotná lokalizace často obnáší porovnání dat ze senzorů s celou mapou.

Schopnost reprezentovat dynamické prosředí znamená dokázat reagovat na změny v prostředí, ve kterém se robot nachází (např. pohybující se lidé a jiné objekty), bez toho, aby byl robot zmaten.

V neposlední řadě k důležitým vlastnostem patří také schopnost detekce a obnovy při selhání lokalizace.

2.2 Problém lokalizace v neznámém prostředí

Předmětem lokalizace v neznámém prostředí je snaha robota zmapovat svoje okolí. Robot se nachází v situaci, kdy je "zapnut"na neznámém místě a musí zjistit, jak vypadá jeho okolí, aby se v něm mohl snadněji pohybovat.

Tento druh lokalizace není předmětem této práce a nebude podrobněji rozepisován. Pro celistvost tématu je zde však uveden.

2.3 Problém lokalizace ve známém prostředí

Též nazývano *problém globální lokalizace* (z angl. global localization problem).[11]

Tento druh lokalizace spočívá v nalezení a udržování informace o poloze robota ve zmapovaném prostředí. Robot se nachází v situaci, kdy je "zapnut"na neznámém místě známého terénu a jeho ukolem je najít svoji polohu na mapě, kterou má k dispozici.

2.3.1 Algoritmy globální lokalizace

Mezi nejjednoduší algoritmy tuto problematiku řešící patří Kalmanův filtr, který však trpí několika nedostatky. Jednak není schopen reprezentovat jiné než Gaussovské rozložení pravděpodobnosti. A jednak není schopen reagovat na dynamické prostředí.

Dalším algoritmem je Markovova lokalizace. Tento algoritmus se již dokáže vypořádat s negaussovským rozložením praveěpodobnosti, avšak stále se nedovede vypořádat s dynamickým prostředím.

Oba uvedené nedostatky úspěšně řeší metoda Monte Carlo, kterou se budeme zabývat v této práci (nejen) v následující kapitole.

Kapitola 3

Monte Carlo lokalizace

V této kapitole bude čtenář podrobněji seznámen s metodou lokalizace Monte Carlo (dále jen MCL).

Nejprve stanovíme několik pojmů, které budeme v následujícím textu používat. *Poloha* robota představuje souřadnice na mapě (dále uvažujme 2-rozměrné souřadnice (x,y); polohu budeme někdy označovat také jako *bod*). *Pozice* robota je jeho poloha a jeho odchylka (tedy odchylka dopředného vektoru robota) od kladné poloosy x mapy (tuto odchylku budeme nazývat také *orientace*). *Vzorek* je dvojice (pozice, váha) představující jistou pravděpodobnost, že se robot nachází na dané pozici, kde váha je hodnota této pravděpodobnosti. *Pohybová jednotka* je trojice (počáteční natočení, posun, koncové natočení), která slouží k relativní (vůči nějaké pozici) reprezentaci obecného pohybu, přičemž nás nezajímá trajektorie pohybu, ale jen počáteční a cílová pozice (jedná se tedy o pohyb z pozice do pozice, ne jen z bodu do bodu).

3.1 Princip metody

V této sekci bude popsán princip lokalizační metody Monte Carlo.

3.1.1 Neformální popis

Principem této metody je vytvoření konečné množiny vzorků rovnoměrně rozprostřených po celé mapě oblasti a se shodnou vahou. Jednotlivé vzorky jsou časem posouvány (tedy mění se pozice, kterou reprezentují) na základě informací z odometrie. A váha vzorků je upravována na základě informací ze senzorů.

Algoritmus probíhá iterativně. V každé iteraci je vždy proveden posun všech vzorků a určení nové váhy všech vzorků (*predikce*). Po provedení těchto dvou věcí přichází na řadu zásadní krok a tím je *převzorkování* (nebo také *korekce*). To obnáší vytvoření nové množiny vzorků na základě původní množiny takovým způsobem, že vzorky z původní množiny, které měly velikou váhu budou v nové množině třeba i několikrát. A vzorky které v původní množině měly nízkou váhu, v nové množině třeba ani nebudou. Tím postupně dojde k tomu, že vzorky neodpovídající skutečnosti jsou odstraněny a vzorky, které mají vysokou pravděpodobnost, že odpovídají skutečnosti, zůstávají.

3.1.2 Zápis v pseudokódu

1. MCL_algorithm(X_{t-1}, u_t, z_t) :
2. $X'_t = X_t = \emptyset$
3. For i = 1 to N:
4. $x^i_t = p(x^i_t | x^i_{t-1}, u_t)$
5. $w^i_t = p(z_t | x^i_t)$
6. $X'_t = X'_t + [x^i_t, w^i_t]$
7. For i = 1 to N:
8. Rozšiř X_t o náhodný vzorek z X'_t s pravděpodobností danou váhou vzorku
9. Return X_t

Funkce výše představuje jednu iteraci algormitmu MCL. X_{t-1} je množina vzorků po předchozí iteraci, u_t jsou informace z odometrie a z_t jsou informace ze senzorů, N je počet vzorků (prvků množiny X_{t-1}). Řádky 4 a 5 představují predikci. Řádky 7 a 8 představují korekci neboli převzorkování. Výstupem je nová množina vzorků.

3.2 Detaily jednotlivých částí algoritmu

V této sekci budou detailněji popsány jednotlivé části algoritmu Monte Carlo lokalizace.

3.2.1 Predikce

Predikce je fází, ve které dochází k posunu (změně pozice) vzorků na základě informací z odometrie. Informací z odometrie je pohybová jednotka. Informace z odometrie nemohou být považovány za přesné a to je potřeba zohlednit. To uděláme tak, že jednotlivé vzorky neposouváme přesně o hodnoty dané odometrií, ale posuneme je o hodnotu mírně ovlivněnou náhodným šumem (pro každý vzorek generujeme samostanou hodnotu šumu v určitém rozmezí).

Dalším úkolem predikce je určení podobnosti jednotlivých vzorků s daty naměřenými senzory robota. Proces určení této podobnosti je závislý na použitých senzorech. Robot má možnost získat informace ze senzorů v místě, kde se nachází. Aby však mohl úspěšně provádět predikci, musí umět také odhadnout jaká data by mu jeho senzory poskytly, kdyby se nacházel na libovolné pozici. Druhá část predikce tak představuje porovnání dat naměřených senzory s daty, které by senzory naměřily na pozicích daných jednotlivými vzorky, a určení jejich podobnosti.

3.2.2 Korekce

Tato fáze má zásadní vliv na funkci celého algoritmu MCL. Rozložení pravděpodobnosti, které bylo zpočátku rovnoměrné po celé mapě, se díky této fázi transformuje na rozložení, které s každým krokem algoritmu stále více odpovídá skutečnosti. Jejím principem je vytvoření nové množiny vzorků, kde má každý vzorek sice stejnou váhu, ale tato množina již neobsahuje ty vzorky, které příliš neodpovídají skutečnému stavu robota. Naopak vzorky, které poměrně dobře odpovídají skutečnosti, jsou duplikovány (některé dokonce několikrát).

Kapitola 4

Návrh aplikace

V této kapitole se čtenář seznámí s návrhem aplikace demonstrující MCL algoritmus na reálném robotovi.

4.1 Požadavky na aplikaci

Základním požadavkem na aplikaci je schopnost lokalizovat robota pomocí MCL algoritmu a průběh MCL algoritmu graficky zobrazovat na obrazovce. Bude tedy potřeba používat grafické knihovny pro zobrazování průběhu algoritmu. Dále komunikovat s robotem - získávat data ze senzorů a informace o jeho pohybu. Protože smyslem Monte Carlo lokalizace je autonomní orientace v prostoru, bude součástí aplikace také algoritmus pro hledání cesty k cíli a robot bude navigován tak, aby tohoto cíle dosáhl bez dalšího zásahu uživatele. Prostřednictvím GUI bude možné také nastavovat různé parametry použitých algoritmů a v neposlední řadě také zvolit mapu.

4.2 Školní robot

Pro aplikaci MCL algoritmu na reálného robota máme k dispozici školního robota vyvíjeného na UITS. Tento robot je osazen čtyřmi koly, řídící deskou Scorpion, třemi sonary, kompasem, akcelerometrem [7]. Vše je propojeno pomocí FITkitu [1]. FITkit společně s Bluetooth adaptérem umožňuje robotovi kominikovat s PC po seriovém portu.

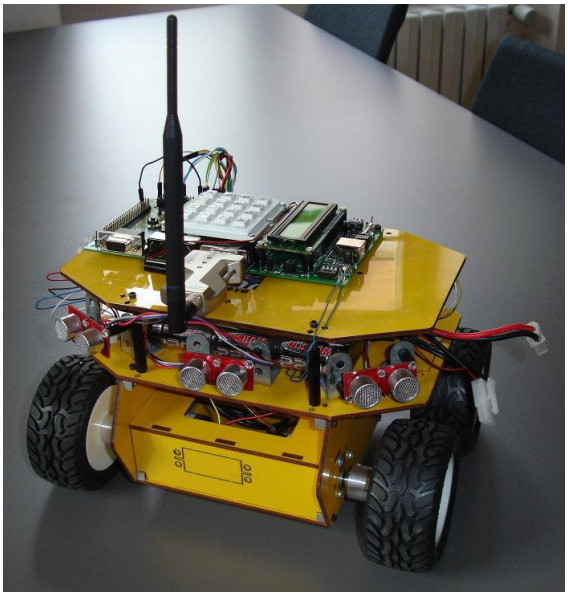
4.2.1 Komunikace

S robotem je možno komunikovat po sériovém portu prostřednictvím Bluetooth adaptéru Handy Port HPS-120. Robot pravidelně posílá na sériový port aktuální hodnoty senzorů robota. A reaguje na příjem řídících znaků po sériovém portu [7].

4.2.2 Ovládání

Ovládání pohybu robota se uskutečňuje prostřednictvím sériového portu. Robot reaguje na příjem některých znaků. V aplikaci použijeme jen tyto:

- 'i' a 'k' - Způsobuje pohyb vpřed (resp. vzad).
- 'j' a 'l' - Vyvolává pohyb vlevo (resp. vpravo). Robot při tom točí koly na jedné straně do leva a na druhé doprava, což způsobuje v ideálním případě otáčení na místě.



Obrázek 4.1: Školní robot vytvořený během práce [7]

Prokluzování a další situace, které mohou vést k tomu, že otáčivý pohyb nebude zcela na místě opět neuvažujeme.

- 'o' - Zastaví robota.

Z tohoto způsobu řízení vyplývá, že (v případě, že se robot pohybuje na vodorovném neprokluzujícím povrchu) se robot buď pohybuje rovně (vpřed nebo vzad) nebo se otáčí na místě nebo stojí.

4.2.3 Odometrie

V pozdější práci jsou k dispozici i informace z odometrie, udávající vzdálenost, kterou ujela jednotlivá kola [9]. Bohužel nedetekují prokluzování a smyk, což nebudeme uvažovat, protože naši aplikaci budeme testovat pouze v prostředí kde k by tomu nemělo docházet. Protože nebudeme uvažovat prokluzování, použijeme údaje jen z jednoho kola a použijeme je pro odhad ujeté vzdálenosti rovně při pohybu vpřed nebo vzad. Nebudeme těchto údajů však využívat pro měření otáčení.

4.2.4 Kompas

Kompas robota umožňuje zjišťovat orientaci robota vůči magnetickému severu Země. Toho lze využít hned v několika oblastech řešení daného problému. Jednak lze zefektivnit MCL algoritmus dopředným odstraněním těch vzorků, které se s kompasem neshodují a tím snížit časové nároky. A jednak toho lze využít při určování odometrických dat při otáčení (tedy při určení úhlu, o který se robot skutečně otočil). Pokud bychom se na informace kompasu však spoléhali příliš, mohlo by dojít k tomu, že informace nebudou správné (např. pokud se robot nachází poblíž zdroje elektromagnetického pole). Takovou situaci však rovněž nebudeme brát v potaz.

4.2.5 Sonary

Sonary jsou rozmístěny na vnějším okraji robota. Jeden přímo vpřed. 45° doleva i doprava od předního sonaru je předolevý a předopravý sonar. Další dva jsou umístěny o 180° oproti dvěma předchozím (tedy zadolevý a zadopravý sonar). Rozsah měření je od 3cm do 6m. Aktuální naměřené hodnoty jsou k dispozici (teoreticky) každých 65ms, což by mělo být naprosto dostačující. Výsledkem měření je vzdálenost v centimetrech. Sonary slouží jako velice důležitý zdroj informací pro MCL algoritmus. Bude však vhodné je využít i v řídícím modulu k detekci neočekávaných překážek při pohybu.

4.3 Model prostředí - formát mapy

Prostředí budeme modelovat jako dvourozměrnou diskrétní mapu volného prostoru a překážek. Vycházíme z toho, že je aplikace určena jen pro pohyb robota v prostředí s vodorovným povrchem. Zároveň je to i vhodné, protože budeme průběh lokalizace zobrazovat na monitoru. Navíc se v praxi ukazuje, že 2D reprezentace je dostačující zejména pro prostory budov, kde budeme naši aplikaci testovat [11]. Nebudeme uvažovat rozsáhlé mapy, kde by bylo třeba dělit mapu na sektory (například jednotlivá patra).

Samotná mapa bude pro jednoduchost uložena jako obrázek v BMP formátu, kde černá reprezentuje překážky a bílá (resp. jakákoliv jiná než černá) reprezentuje volný prostor. Pro vytváření a úpravy map tak není potřeba vytvářet žádné zvláštní rozhraní, ale stačí použít libovolný dostupný bitmapový editor. Měřítko mapy však musí být explicitně uvedeno (možno nastavit prostřednictvím GUI).

Zároveň při načítání mapy z obrázku, rozšíříme překážky o určitou malou vzdálenost pro účel hledání cesty. To proto, aby se robot nepokoušel plánovat cestu vedoucí příliš blízko překážek, což by mohlo vést k fyzickému kontaktu robota s překážkou a případnému poškození robota nebo překážky. Pro simulaci senzorických měření však budou uvažovány původní rozměry překážek.

4.4 Modul MCL

Algoritmus MCL pracuje obecně bez nutnosti znát jakými senzory a pohybovými součástkami je robot vybaven. Protože však potřebuje vstupy od těchto dvou součástí robota, je nutné dohodnout nějaké společné rozhraní. Pro tento účel budou zavedeny dva moduly, které toto rozhraní zajistí. Prvním modulem je tedy modul odometrie, tím druhým je modul senzorů.

4.4.1 Generátor náhodných čísel

Z principu algoritmu MCL, vyplývá nutnost použít generátor náhodných čísel. Implementace generátoru skutečně náhodných čísel je velice obtížná a v praxi se spíše používají generátory pseudo-náhodných čísel. Jednoduchý (tzv. kongruentní) generátor pseudo-náhodných čísel je součástí jazyka C. Je však známo, že tyto jednoduché generátory mají několik nedostatků, které v případě algoritmu MCL nelze zanedbat [8]. Mezi tyto nedostatky patří krátká perioda generátoru a také vlastnost, kdy při použití generátoru pro generování souřadnic dochází k jakési pravidelnosti, která je pro skutečně náhodné jevy nežádoucí.

Z těchto důvodů při implementaci využijeme generátor Mersenne-Twister, který má dostatečně dlouhou periodu ($2^{19937} - 1$) a nejeví pravidelnost při generování souřadnic až

do 623 dimenzí [12].

4.4.2 Modul odometrie

Tento modul slouží k předávání informací o relativní změně pozice robota. Tato relativní změna přísluší vždy nějakému časovému intervalu. Během toho časového úseku se robot pohybuje po obecné trajektorii. Pro naše účely však není nutné znát celou tuto trajektorii.

Vystačíme s trojicí údajů - počáteční natočení, přímý posun a koncové natočení. Pomocí těchto tří údajů jsme schopni popsat pohyb robota z libovolné pozice do libovolné jiné pozice - tuto trojici budeme dále nazývat *pohybová jednotka*.

Postačující tedy je, aby bylo možné v libovolný moment získat informaci o změně pozice robota za dobu od posledního požadavku na tuto informaci. Při simulaci je možné tuto informaci nahradit přímo pohybem, o který zažádal modul, který (virtuálního) robota řídí. U robota reálného je nutné tuto informaci vypočítat z údajů z robotových akcelerometrů. Jednak protože je možné, že za dobu mezi dvěma požadavky robot nestihne požadovaný pohyb vykonat. A jednak protože je také možné, že požadovaný pohyb není vůbec fyzicky možný.

Dostupné informace od robota

Robot nám pro účely odometrických měření poskytuje následující údaje:

- Posun vpřed / vzad - prostředníctvím akcelerometrů
- Rotace vlevo / vpravo - prostřednictvím kompasu

Oba případy lze převést na pohybovou jednotku s jen jednou nenulovou položkou. Zdá se tak, že nepotřebujeme tak obecnou (relativně) reprezentaci pohybu. Protože ale budeme potřebovat pohybové jednotky sčítat, může vzniknout libovolný pohyb (pohybová jednotka) obsahující všechny tři položky nenulové.

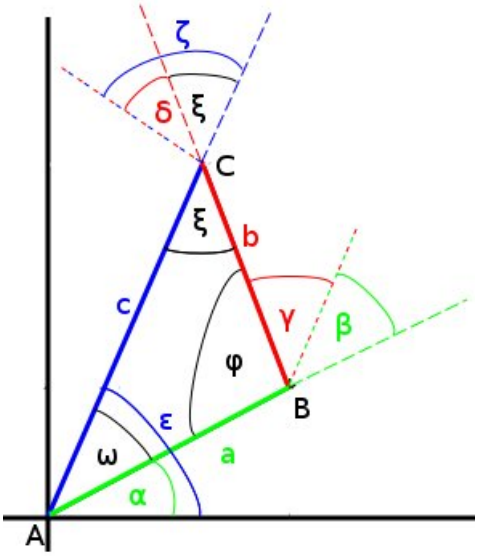
Na obrázku 4.2 to můžeme vidět v obecném případě. Pokud robot nahlásí dva pohyby (z A do B, a z B do C), ale k vyzvednutí pohybu dojde jen v A a C, bude vyzvednut celý pohyb (z A do C). Orientace robota před zahájením pohybu byla shodná s kladným směrem osy x (tedy 0). Po prvním pohybu je orientace $\alpha + \beta$. Po druhém pohybu $\alpha + \beta + \gamma + \delta$.

Sčítání pohybových jednotek

Jelikož reálný robot posílá informace velice často, je více než pravděpodobné, že během jedné iterace MCL algoritmu, přijde několik informací odometrie po sobě. MCL algoritmus potřebuje zjistit informace odometrie každou iteraci právě jednou. Je tedy nutné zajistit, aby modul odometrie, mezi jednotliými vyzvednutími ujetého pohybu informaci o tomto pohybu sčítal, nikoli aby jen předal poslední informaci obdrženou od robota. Je tedy nutné určit způsob jak sečíst dvě pohybové jednotky.

To si ukážeme s pomocí obrázku 4.2

- (α, a, β) - první pohybová jednotka - pohyb z bodu A do B
- (γ, b, δ) - druhá pohybová jednotka - pohyb z bodu B do C
- (ϵ, c, ζ) - výsledná pohybová jednotka - pohyb z bodu A do C



Obrázek 4.2: Výpočet součtu dvou pohybových jednotek

Platí, že

$$\phi = \pi - \beta - \gamma \tag{4.1}$$

Dle cosinovy věty určíme výsledný posun:

$$c = \sqrt{a^2 + b^2 - 2ab \times cos\phi} \tag{4.2}$$

Pomocí sinovy věty určíme úhel ω

$$\omega = arcsin(\frac{b \times sin\phi}{c}) \tag{4.3}$$

Nyní můžeme určit oba úhly ϵ i ζ

$$\epsilon = \alpha + \omega \tag{4.4}$$

$$\zeta = \delta + \pi - \phi - \omega \tag{4.5}$$

Pokud by c bylo 0, potom je výsledný posun nulový, nemění se tedy poloha, ale jen orientace. A výsledná pohybová jednotka se pak vypočítá takto:

$$\epsilon = \alpha + \beta + \gamma + \delta \tag{4.6}$$

$$c = \zeta = 0 \tag{4.7}$$

Když máme mechanismus, jak sečíst dvě pohybové jednotky, stačí nám při vyzvednutí pohybu MCL modulem vnitřní čítač pohybu vynulovat a než o hodnotu zažádá znovu tak přičítat každou informaci o pohybu obdrženou od robota.

4.4.3 Modul senzorů

Tento modul slouží k porovnání informací ze senzorů s jednotlivými vzorky. Jeho součástí tedy musí být mechanizmus simulace senzorových dat podle polohy vzorku. A dále je za potřebí model, který porovná simulovaná měření s těmi skutečnými a určí jakou měrou jsou si tyto podobné. Proces porovnání zde má velice významnou funkci a musí zohlednit fakt, že mapa, kterou má robot k dispozici není naprosto přesným obrazem reality. Navíc je nanejvýš pravděpodobné, že vzorek, který by přesně odpovídal skutečné pozici robota, ani neexistuje.

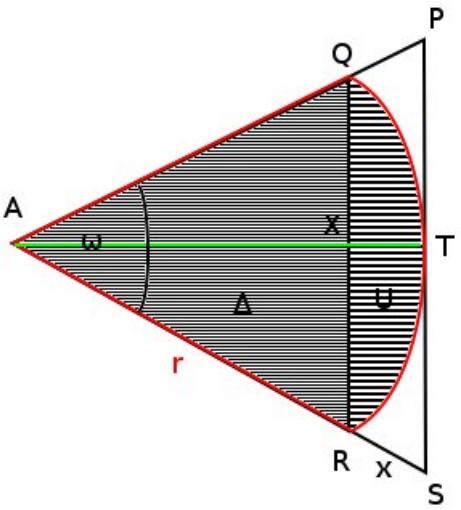
Protože v našem případě bude algoritmus testován na skutečném robotovi, který bude vybaven senzory měřící vzdálenosti k nejbližší stěně (či jiné odrazové ploše), uvedeme zde model, který takovéto senzory modeluje (tento model se v anglické literatuře nazývá *Range Finders Model*) [11].

Modul senzorů bude zároveň poskytovat i informaci o orientaci robota vůči magnetickému poli Země pomocí kompasu. Ačkoliv to není nutně potřebné pro MCL algoritmus, využijeme toho k snížení počtu vzorků

4.4.4 Simulace senozorových měření

V této podkapitole si vysvětlíme jakým způsobem budeme počítat hodnoty, jaké by senzory naměřili, kdyby se robot nacházel na libovolné (zvolené pozici). Připoměňmě, že toto je potřeba pro určení podobnosti vzorku se skutečností.

Již jsme zmínili dříve, že v případě snímačů vzdálenosti, může být toto prováděno metodou *ray casting*. To je ovšem řešení vhodné pro laserové snímače. Modul senzorů sloužící pro řízení našeho robota bude využívat jinou metodu. To z toho důvodu, že robot, kterého budeme řídit bude vybaven radarovými snímači, které neměří vzdálenost k nejbližšímu objektu po přímce, ale po úzké kruhové výseči.



Obrázek 4.3: Princip fungování algoritmu simulovaných sonarových měření

K tomu jsem vytvořil algorimus založený na Bresenhamově algoritmu pro rasterizaci úsečky [5]. Jeho principem je hledání nejbližšího objektu tak, že se současně prochází okrajové polopřímky zorného pole radaru. Bod po bodu směrem od radaru, aktuální bod jedné okrajové poloppřímky je spojen s aktuálním bodem druhé okrajové polopřímky další úsečkou a na této spojnici hledáme překážku. Pokud je překážka nalezena vrátíme vzdálenost k této překážce. Jinak se pokračuje dalšími body na okrajových polopřímkách, tak dlouho dokud není nalezena překážka nebo není dosaženo maximální vzdálenosti (dáno dosahem radaru).

Princip je zobrazen na obrázku 4.3. Na obrázku vidíme zorné pole sonaru, šířka zorného pole je ω. r představuje dosah sonaru. Plocha kruhové výseše ARTQ tvoří celý prostor, ve kterém je sonar schopen detekovat překážky (chyby měření, ke kterým dochází u skutečných senzorů při simulaci uměle nevytváříme). Algoritmus tedy prochází oblast po úsečkách spojujících body na okrajových polopřímkách (AQ a AR) tak dlouho, než narazí na překážku, nebo dosáhne maximální vzdálenosti. Pokud nenalezne překážku, vrátí maxímalní vzdálenost. To ovšem vede k tomu, že je prozkoumána jen plocha trojúhelníka ARQ (tedy plocha Δ). Celá oblast kruhové úseče RTQ (plocha U) však zůstává neprozkoumána na případné překážky, které jsou ve skutečnosti blíž než je maximální dosah sonaru.

Řešením tohoto problému je nezastavit při dosažení maximálního dosahu (tedy při dosažení bodu R, resp. Q), ale místo toho pokračovat až do bodu S (resp. P). Zde ovšem už při procházení spojnice, pokud najdeme překážku, vratíme vzdálenost k ní, jen v případě, že je tato vzdálenost menší nebo rovna maximální vzdálenosti. Jinak překážku ignorujeme a pokračujeme v prohledávání.

Algoritmus tak sice projde celý povrch trojúhelníka ASP, který obsahuje i body mimo dosah radaru, ale na druhou stranu už nebude docházet k tomu, že vypočtená hodnota bude maximální i když by neměla. Musíme však odvodit vztah pro velikost úsečky RS. Podle toho poznáme, jak daleko za dosah radaru máme ještě pokračovat.

$$|RS| = x \tag{4.8}$$

$$|AT| = |AR| = r \tag{4.9}$$

$$|AS| = |AR| + |RS| = r + x \tag{4.10}$$

V pravoúhlém trojúhelníku ATS platí:

$$cos\frac{\omega}{2} = \frac{|AT|}{|AS|} = \frac{r}{r+x} \tag{4.11}$$

tedy

$$x = r(\frac{1}{cos\frac{\omega}{2}} - 1) \tag{4.12}$$

Celková vzdálenost, do které se má prohledávání uskutečnit je tedy dána takto:

$$d = r + x = r + \frac{r}{cos\frac{\omega}{2}} - r = \frac{r}{cos\frac{\omega}{2}} \tag{4.13}$$

Můžeme si všimnout, že k tomuto vztahu lze dospět i na základě podobnosti trojúhelníků AXR a ATS.

4.4.5 Pravděpodobnostní model snímačů vzdálenosti

Následující model vychází z modelu uvedeného v publikaci *Probabilistic Robotics* (kapitola Beam Models of Range Finders) [11].

Snímače vzdálenosti měří vzdálenost k nejbližší odrazivé ploše. Jak jsme zmínili, jen těžko lze předpokládat, že by nějaký vzorek ležel přesně tam ,kde se ve skutečnosti nachází robot. Proto je každému vzroku přiřazováná pravděpodobnost shody se skutečností (dále jen *podobnost*, z angl. *likelyhood*). Reálné senzory jsou navíc zatíženy chybou měření, která může mít několik příčin. Navíc (především v dynamickém prostředí) nemusí mapa zcela odpovídat skutečnosti. To vše je potřeba v tomto modelu zohlednit. Jednotlivé faktory, které je třeba zohlednit jsou:

- Nepřesnosti měření - Je celkem běžné, že senzory vlivem atmosferických a jiných faktorů naměří hodnotu o trochu odlišnou od skutečnosti. Tímto se zároveň zohledňuje fakt, že porovnávaný vzorek jen vyjímečně bude na stejné pozici jako je reálná pozice robota.
- Dynamičnost prostředí - V dynamickém prostředí se může stát, že se mezi snímačem a odrazivou plochou, na které by snímač změřil vzdálenost, objeví objekt, který není na mapě a způsobí, že naměřená hodnota je nižší než by se očekávalo.
- Chyba detekce objektu - Někdy se stane, že přestože snímacímu paprsku leží v cestě nějaká překážka, snímač nezachytí vracející se paprsek a dojde tak k tomu, že si snímač myslí, že naměřil vzdálenost vetší nebo rovnu svému maximálnímu dosahu.
- Náhodné chyby - Někdy se stane (příčiny mohou být různé), že snímač naměří hodnotu, která vypadá naprosto náhodně a není patrný žádný vztah ke skutečné hodnotě, která by zde měla být naměřena.

K určení podobnosti vzorku se skutečností se využívá funkce (jde vlastně o funkci hustoty pravděpodobnosti), jejímiž vstupy jsou naměřená vzdálenost a očekávaná vzdálenost a výstupem je podobnost vzorku se skutečností. Tato funkce je složena z několika dílčích funkcí a každá z těchto dílčích funkcí zohledňuje jeden z faktorů uvedených výše.

Než si přiblížíme jednotlivé funkce, uveďmě některé skutečnosti, které k tomu použijeme. Každý snímač má nějaký omezený dosah. Označme tento maximální dosah z_{max} a předpokládejme, že v čase t snímač k naměřil vzdálenost z_t^k. Vzdálenost, kterou by snímač měl naměřit na pozici vzorku označíme z_t^{k*} (poznamenejme, že např. v případě laserových snímačů je hodnota z_t^{k*} počítána metodou *raycasting*).

Přistupme tedy k popisu jednotlivých funkcí, které utváří výslednou hustotu pravděpodobnosti, že vzorek odpovídá skutečnosti.

Nepřesnost měření se reprezentuje pomocí normálního (Gaussova) rozložení hustoty pravděpodobnosti se střední hodnotou z_t^{k*} a rozptylem σ_{hit}^2 (vhodně zvolená konstanta).

$$p_{hit}(z_t^k) = \begin{cases} \eta N(z_t^k, z_t^{k*}, \sigma_{hit}^2) & \text{if } 0 \leq z_t^k \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{4.14}$$

$$N(z_t^k, z_t^{k*}, \sigma_{hit}^2) = \frac{1}{\sqrt{2\pi\sigma_{hit}^2}} e^{-\frac{1}{2}\frac{(z_t^k - z_t^{k*})^2}{\sigma_{hit}^2}} \tag{4.15}$$

Normalizační faktor η se vypočítá následovně

$$\eta = \frac{1}{\int_0^{z_{max}} N(z_t^k, z_t^{k*}, \sigma_{hit}^2) dz_t^k} \tag{4.16}$$

Dynamičnost prostředí je znázorněna pomocí exponenciálního rozložení.

$$p_{short}(z_t^k) = \begin{cases} \eta \lambda_{short} e^{-\lambda_{short} z_t^k} & \text{if } 0 \leq z_t^k \leq z_t^{k*} \\ 0 & \text{otherwise} \end{cases} \tag{4.17}$$

kde η je

$$\eta = \frac{1}{\int_0^{z_t^{k*}} \lambda_{short} e^{-\lambda_{short} z_t^k} dz_t^k} = \frac{1}{1 - e^{-\lambda_{short} z_t^{k*}}} \tag{4.18}$$

Chybné detekce jsou modelovány pomocí zvýšené pravděpodobnosti pro $z_t^k = z_{max}$.

$$p_{max}(z_t^k) = \begin{cases} 1 & \text{if } z_t^k = z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{4.19}$$

Náhodné chyby jsou reprezentovány rovnoměrným rozložením pravděpodobnosti přes celý interval $< 0, z_{max} >$.

$$p_{rand}(z_t^k) = \begin{cases} \frac{1}{z_{max}} & \text{if } 0 \leq z_t^k < z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{4.20}$$

Podobnost je tedy vyjádřena funkcí:

$$p(z_t^k) = z_{hit} * p_{hit}(z_t^k) + z_{short} * p_{short}(z_t^k) + z_{max} * p_{max}(z_t^k) + z_{rand} * p_{rand}(z_t^k) \tag{4.21}$$

kde z_{hit}, z_{short}, z_{max} a z_{rand} jsou vhodně zvolené konstanty. A p_{hit}, p_{short}, p_{max} a p_{rand} jsou funkce zohledňující jednotlivé chybové faktory uvedené výše.

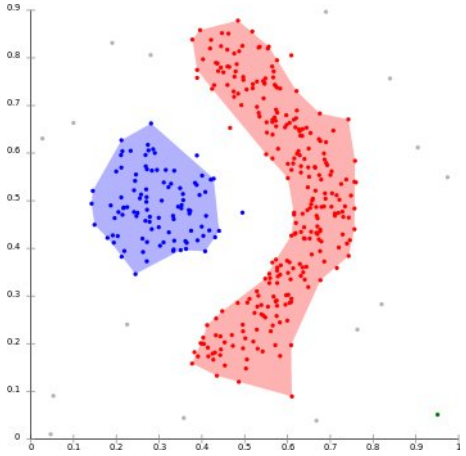
To jsme ovšem spočítali pouze podobnost k-tého snímače. Ovšem ve skutečnosti obvykle robot není vybaven jen jedním snímačem. Výsledná podobnost pro všechny snímače je dána součinem podobností jednotlivých snímačů (N je počet snímačů):

$$p(z_t) = \prod_{k=0}^{N} p(z_t^k) \tag{4.22}$$

4.4.6 Odhad skutečné pozice

Téma, které by rozhodně nemělo zůstat opomenuto, je způsob jakým vlastně určíme skutečnou pozici robota. Výstupem MCL algoritmu je množina vzorků, což mohou být stovky až tisíce prvků.

Jednou z možností je spočítat vážený průměr jednotlivých vzorků podle jejich váhy. Toto řešení má však několik nedostatků. Jednak není schopno rozlišit, kdy se MCL algoritmus nachází ve stavu, kdy ještě vůbec neví, kde se robot nachází a kdy už je pozice známá. A jednak pokud odhad ještě není správný, tak metoda selhává. V případě, že je odhad zatím velice nepřesný, dostaneme váženým průměrem vlastně jen relativně náhodnou pozici na



Obrázek 4.4: Vizualizace dvou shluků rozlišených pomocí DBSCAN algortmu použitého pro určení odhadu pozice robota (obrázek převzat z [2])

mapě. V případě, že už se robot celkem zorientoval, ale na mapě existuje několik podobných lokací a robot se nachází v jedné z nich, tato metoda zcela selhává, protože odhadne pozici v jejich těžišti. Jedině v případě dobrého odhadu jediné správné pozice tato metoda uspěje.

Docházíme tedy k nutnosti hledat algoritmus, který dokaže v množině vzorků vyhledat takové, které si jsou (co do pozice) blízké. Existuje mnoho algoritmů řešící tento problém (anglicky nazývany *means clustering*). Pro naše účely však potřebujeme takový, který umožňuje hladání předem neznámého počtu shluků. Vhodným algoritmem je například algoritmus DBSCAN [6]. Po nalezení disjunktních shluků vzorků můžeme vybrat ten, jehož váha (tedy součet vah jeho vzorků) je největší. A teprve u vzorků tohoto shluku spočítáme vážený průměr jeho vzorků a tím dostaneme odhad pozice robota. Znázornění principu DBSCAN algoritmu je zobrazeno na obrázku 4.4.

4.4.7 Optimalizace

Zde najdete shrnutí optimalizací, z nichž některé již byli zmíněny.

- Vyloučení vzorků, u kterých je zřejmé že nemohou odpovídat skutečnosti (jedná se zejména o vzorky mimo mapu nebo nacházející se ve zdi).
- Možnost snížit počet vzorků po zlokalizování. Lze nastavit při kolika shlucích má začít snižování počtu vzorků, jak rychle má snižování probíhat. A také minimální počet vzorků.
- Samotný algoritmus DBSCAN byl rozšířen o možnost předčasného ukončení v případě nalezení příliš velkého počtu shluků (což signalizuje, že odhadnutá pozice není správná a tudíž nás výsledek ani nebude zajímat).
- Pro práci se školním robotem, který disponuje kompasem, byla navržena optimalizace MCL algoritmu, která tohoto kompasu využívá. Optimalizace taková, že díky znalosti orientace robota (vůči magnetickému severu Země) umožní odstranit ty vzorky, které této orientaci neodpovídají (s jistou povolenou odchylkou), čímž se sníží počet vzorků

se kterými MCL algoritmus pracuje a tím je dosaženo výrazného zrychlení jednotlivých iterací algoritmu (časová složitost MCL algoritmu je silně závislá na použité metodě simulace senzorických dat, ale obecně je složitější než lineární [11]). Poznamenejme, že aby algoritmus v takovém případě pracoval správně, musí být mapa prostředí správně orientovaná (tedy sever nahoře).

4.5 Hledání cesty

Protože lokalizace robota má být autonomní, je vhodné připojit také modul, který bude samostatně volit cestu k cíly. Uživatel jen nastaví tento cíl. K řešení tohoto problému jsem se rozhodnul použít algoritmus D* [10], který je upravenou verzí algoritmu A*, určený pro použití v podobných situacích (tedy v situacích, kdy je potřeba cestu hledat opakovaně a v reálném čase). Jeho hlavní výhodou je, že dokud se mění jen startovní pozice a ne cíl cesty, není potřeba znovu provádět výpočet (alepoň ne celý). Navíc tento algoritmus má prostředky jak prostřednictvím senzorů robota zohlednit překážky, které nejsou zaneseny v mapě.

Vstup modulu hledání cesty je tedy startovní a cílová pozice. Cíl cesty zvolí manuálně uživatel, a jako start cesty se použije aktuální odhad pozice robota. Připomeňme však, že MCL algoritmus se může nacházet ve stavu kdy ještě odhad pozice robota není správný. Řekli jsme, že tuto situaci dokážeme rozpoznat (viz. kapitola Odhad skutečné pozice). V takové situaci však není vhodné tento odhad použít k určení startovní pozice pro hledání cesty. Proto v tomto případě budeme robota navigovat náhodně se zohledněním údajů ze senzorů robota (nebudeme robota nutit vrážet do zdi).

Rozlišujeme tedy dva druhy rozhodování o dalším pohybu na základě důvěryhodnosti odhadnuté pozice robota. Jako míru důvěryhodnosti stanovíme počet nalezených shluků.

Náhodný pohyb se uplatní, pokud nejsou nalezeny žádné shluky, nebo je nalezeno více než daná hodnota (lze nastavit v GUI), pohyb se bude volit náhodně. Zde je vhodná hodnota nízké číslo (např. 1). Náhodný pohyb je řízen následujícími pravidly:

- Pokud je před robotem volno, jeď rovně.
- Pokud je před robotem překážka, zkotroluj boční sensory a:
  - pokud je volno před levým sonarem, otoč se v jeho směru
  - pokud je volno jen přd pravým otoč se v jeho směru
  - pokud není volno ani před jedním otoč se vlevo
- Další pohyb se zvolí při dalším průchodu.

Účelný pohyb se uplatní, když je nalezeno odpovídající množství shluků. Je vybrán ten shluk jehož celková váha (součet vah vzroků v něm obsažených) je největší a použit jako startovní pozice pro vyhledávání cesty D* algoritmem.

Po vyhledání cesty máme k dispozici seznam bodů, které označují cestu k cíli. Není nutně těmito body projet, ale použijí se k řízení "po úsecích".

Ještě poznamenejme, že by robot měl oznámit dosažení cíle. Nelze však očekávat, že dosáhne přesně zadaného cílového bodu, ale raději budeme detekovat zda robot dorazi do určitého blízkého okolí cílové polohy.

4.6 Řízení robota

Jako příkazy pro řízení robota budeme využívat pohybové jednotky, jak byly definovány v rámci odometrie (natočení, přímy posun, natočení). Pro spolehlivé navigování robota je potřeba zajistit, aby robot reagoval na jednotlivé příkazy požadovaným způsobem. K tomu vytvoříme samostatně pracující modul, který zajistí správné vykonání požadovaných pohybů.

Navíc výstup vlastně vytváří model odometrie pro použití MCL algoritmem. Dá se tedy říci, že kromě toho, že tento modul bude robota řídit, bude zároveň zprostředkovávat data odometrie a data senzorů pro použití MCL algoritmem.

Jako vstup tedy použijeme krátký úsek nalezené cesty (viz. hledání cesty) začínající na aktuální (odhadnuté) pozici robota. Tento úsek převedeme na pohybovou jednotku. A tu předáme jako příkaz k pohybu. Řídící modul zajistí odeslání patřičných signálů robotovi po seriovém portu. Průběžně bude řídící signály upravovat v závislosti na příchozích informacích z akcelerometru, tak aby se co nejvíce přiblížil požadovanému pohybu. Vypočítaný pohyb bude průběžně ukládat, až do vyzvednutí modulem MCL.

Délku úseku cesty použité jako vstup je vhodné určit takovou, aby robot během jedné iterace MCL algoritmu (interval mezi dvěma vyzvednutími vypočítaného pohybu) stihl přibližně akorát požadovaný pohyb provést (když budeme volit příliš dlouhé pohyby, bude program zbytečně neefektivní, krátké úseky budou způsobovat pozastavování robota).

4.7 Grafické uživatelské rozhraní

K vytvoření GUI aplikace využijeme multiplatformní knihovnu wxWidgets. [3] Hlavní okno aplikace bude obsahovat dvě části, v jedné se bude zobrazovat zvolená mapa a průběh algoritmu. Ve druhé bude možno volit různá nastavení, vybírat mapu, navazovat spojení s robotem a podobně. K vytváření samotných dialogů a oken je k dispozici aplikace wxformbuilder [4], která umožňuje vygenerovat požadované třídy v C++ pro použití s wxWidgets.

Kapitola 5

Implementace

Navržená aplikac je implementována v jazyce C++ s využitím grafické knihovny wxWidgets [3]. Celou aplikaci utváří tři hlavní moduly, které jsou implementovány jako samostatná vlákna:

- Grafické vlákno - zachytává uživatelské požadavky a zobrazuje průběh lokalizace
- Výpočetní vlákno - provádí výpočetně náročné operace
- Kominikační vlákno - zajišťuje spojení s robotem.

5.1 Výpočetní vlákno

V této kapitole si popíšeme výpočetní vlákno programu. Implementace vlákna najdete v souboru *mcl_thread.h*. Výpočetní vlákno zajišťuji tři hlavní funkce:

- Lokalizace pomocí lokalizačního algoritmu Monte Carlo.
- Hledání cesty pomocí D* algoritmu
- Výběr pokynů pro řízení robota.

5.1.1 Modul MCL

Všechny součásti tohoto algoritmu jsem implmentovány jako součást jmenného prostoru `mcl`. Jeho součástí je několik definic.

- `class mcl::angle` - třída obalující typ double, zajišťuje normalizaci hodnoty do intervalu $< -\pi, \pi >$
- `class mcl::point` - reprezentuje polohu
- `class mcl::position` - poloha rozšířená o natočení, tedy pozice
- `class mcl::motion` - reprezentuje datovou strukturu odometrických dat
- `class mcl::unit` - zapouzdřuje algoritmus MCL se všemi proměnnými
- `class mcl::odometry` - představuje rozhraní mezi modulem MCL a modulem odometrie
- `class mcl::perception` - utváří rozhraní mezi modulem MCL a modulem senzorů

Instance třídy `mcl::unit` provádí samotnou lokalizaci. Při její inicializaci je třeba jí předat konkrétní instance modulů odometrie a senzorů. Pro implementaci konkrétního odometrického modelu resp. modelu senzorů stačí odvodit novou třídu od `mcl::odometry` resp. `mcl::perception` a doimplementovat jejich virtuální metody. Tímto je splněna podmínka obecnosti, protože je tím umožněno využít jedinou implementaci MCL algoritmu (`mcl::unit`) a aplikovat ji na libovolné senzory.

Definice modulu MCL se nacházejí v souborech *mcl.h* a *mcl_types.h*.

Genrátor náhodných čísel

V aplikaci použijeme implementaci generátoru Mersenne-Twister, která je volně ke stažení na internetu [12]. Kopie hlavičkového souboru generátoru se nachází ve složce *improper*.

Odhad skutečné pozice robota

Způsob jakým je z množiny vzorků MCL algoritmu odvozena odhadnutá pozice robota, není přímo součástí MCL algoritmu. V naší aplikaci je to řešeno (jak již bylo zmíněno v kapitole Návrh aplikace) pomocí algoritmu DBSCAN [6]. Implementace tohoto algoritmu je však oddělena, naleznete ji v souboru *dbscan.h*.

5.1.2 Simulace senzorů

Definice modulu senzorů najdete s souboru *sim_sensors.h*. Implementace Bresenhamova algoritmu je odděleně v souboru *line.h* a implementace samotné simulace radarových měření naleznete v souboru *raycasting.cc* (ačkoliv se vůbec nejedná o raycasting, je to pozůstatek z inkrementální fáze vývoje aplikace)

5.1.3 Mapa prostředí

Implementaci mapy je možné nalézt v souboru *map.h*. Jednak zde naleznete třídu `MapPoint`, která reprezentuje jednu buňku mapy (odpovídá jednomu pixelu zdrojového obrazu). A dále třídu `RobotMap`, která zaštiťuje celou mapu (tedy pole těchto buněk). Nabízí funkce jako načtení z obrázku, změnu měřítka, přístup k jednotlivýcm buňkám, a další.

5.1.4 Hledání cesty

Algoritmus hledání cesty (D*) je implementován v souboru *dstar.h* (resp. *dstar.cc*) jako třída `DSTAR`. Protože využívá podobné datové struktury jako MCL algoritmus (tedy pole buněk) a protože, ačkoliv není přímo součástí lokalizace samotné, hledání cesty a MCL budou použity vždy současně, jsou informace potřebné algoritmem D* vloženy přímo do načtené mapy (tedy jsou součástí definice třídy `MapPoint` (viz. kapitola implementace - mapa prostředí). Třída `DSTAR` tak vlastně tvoří jakousi nadstavbu nad třídou `RobotMap`.

5.2 Komunikční vlákno

V této sekci si popíšeme kominukační vlákno. Jeho implementaci najdete v souboru *comm_thread.h* a *comm_thread.cc*. Toto vlákno má tři funkce:

- Zajišťuje komunikaci s robotem v obou směrech

- Implmentuje rozhraní odometrie a senzorů pro modul MCL
- Zprostředkoovává řízení robota

5.2.1 Sériová komunikace

Při implementaci komunikace s robotem se vyskytlo několik problémů, jejichž řešení si zde popíšeme.

Prvním problémem byl fakt, že software ve školním robotovi již není ten ze kterého se vycházelo. Takže bylo potřeba vyhledat vhodný aktuální software. Takový software byl k dispozici v práci Stanislava Sojky [9]. Rozdílem oproti původní implementaci [7] je změna formátu posílaných dat (informací ze senzorů, akcelerometrů,...) a ve způsobu ovládání. Formát posílaných dat není žádnou překážkou, stačí tento formát dodržet. Co se týče způsobu ovládání, tak zde bylo zapotřebí nepatrného zásahu. V implementaci vysavače totiž bylo nahrazeno ovládání přes seriový port ovládáním pomocí joysticku. Kód umoňující ovládání přes seriové rozhraní byl však pouze zakomentován, takže jej stačilo odkomentovat.

Dalším rozdílem v implementaci je fakt, že samotný hardware robota se dočkal také několika změn. Ovládací deska robota (Scorpion) byla od roku 2008 vyměněna za jinou, což ovšem mělo vliv na software robota, nikolv na naši aplikaci. Dále byl robot rozšířen o další dva sonary (takže nyní je vybaven celkem pěti oproti původním třem).

Posledním problémem, který se objevil je skutečnost, že Bluetooth adaptér použitý pro komunikaci s robotem disponuje pouze ovladači pro Windows. Aplikace je sice od začátku vyvíjena pomocí multiplatformní knihovny wxWidgets [3], takže by stačilo testy s robotem provádět ve Windows, ač vývoj aplikace probíhal pod linuxem. Bohužel ale knihovna wxWidgets nedisponuje prostředky pro seriovou komunikaci. Takže bylo potřeba komunikační modul implementovat pod Windows, čimž se aplikace stala nemultiplatofrmní. Nicméně vetšina zdrojových kódů je stále univerzální a tak by se dala využít například jako aplikace sloužící jen k simulaci MCL algoritmu (bez reálného robota).

Implementace funkcí použitých pro sériovou komunikaci najdete v souborech *serial.h* a *serial.cpp*.

5.2.2 Zpracování informací od robota

Třída `CommThread`, která je implementací komunikačního vlákna tedy zajišťuje rozhraní jak senzorů tak odometrie pro modul MCL.

Data senzorů

Údaje ze senzorů robota jsou tedy (společně s údaji z dalších robotových součástí) neustále posílány po sériovém portu. Komunikační modul data neustále čte a ukládá. Tím je zajištěno, že v libovolný okamžik budou k dispozici nejaktuálnější data senzorů.

Odometrie

Odometrie je zajištěna díky robotvým vestavěným akcelerometrům a díky kompasu. Díky mechanizmu sčítání pohybových jednotek uvedeného v návrhu je umožněno v libovolný okamžik obdržet nastřádaný pohyb od posledního vyzvednutí těchto dat. Přestože sčítání pohybových jednotek zabírá vláknu čas a vlákno tak nemůže v tomto čase komunikovat s robotem, není možné přesunout tento úkol do výpočetního vlákna, protože to může (a velice pravděpodobně také bude) v daný moment zaměstnáno jiným výpočtem. Na druhou

stranu, tento výpočet není natolik složitý, aby způsoboval ztráty dat přicházejících od robota v důsledku přetečení příchozího bufferu.

5.2.3 Řízení robota

K řízení robota slouží rovněž rozhraní sériového portu. Robot reaguje na zápis řídících znaků, které způsobují pohyb vpřed, vzad, rotace do stran a zastavení [7]. Narozdíl od příchozích dat, které chodí (relativně) pravidelně, řídící signály je potřeba odesílat jen někdy.

Ovládání pohybu

Komunikační vlákno má k dispozici buffer požadavku na pohyb. Pokud je požadován nějaký pohyb (to si vyžádá vlákno modulu MCL), komunikační modul před dalším čtení dat od robota, odešle adekvátní požadavek na pohyb.

Požadavek na pohyb je buď požadavek na otočení nebo na rovný pohyb kupředu o určitou hodnotu (úhel resp. vzdálenost). Robot však zná příkazy jen pro zahájení pohybu, nikoliv pro zastavení po určitém úhlu (resp. vzdálenosti). Proto s příchozími daty postupně zkracujeme požadavek na pohyb až dosáhne příliš nízké hodnoty (nelze kontrolovat přímo na nulu, protože je pravděpodobné, že přesně této hodnoty nikdy nebude dosaženo). Jakmile této hodnoty dosáhne, odešle se požadavek k zastavení robota. Pokud přijde další požadavek na pohyb ještě dříve než je první pohyb dokončen, bude neprovedený pohyb ignorován a zahájí se vykonávání nového požadavku. Naproti tomu, pokud bude první požadavek dokončen dříve než přijde požadavek nový, robot zůstane po tuto dobu stát.

Detekce hrozících kolizí

Protože nelze zajistit, že nebude požádáno o pohyb, který by vedl ke kolizi s překážkou, je řídíci automat rozšířen tak, že detekuje překážky v bezprostřední blízkosti (pomocí dat ze sonarů) a v případě hrozící kolize odešle požadavek na zastavení. Jelikož ale u sonarů někdy dochází k chybným měřením (a při testech se ukázalo, že docela často), což vedlo k bezdůvodnému pozastavování robota. Proto byl tento mechanizmus upraven tak, aby vyslal požadavek k zastavení až po několika kolizních detekcích po sobě. K určení vzdálenosti která je kolizní vycházíme z maximální rychlosti robota a reakční doby (tedy doby, za kterou jsme schopni, od zjištění kolizního stavu, zastavit). V případě, že požadavek na zastavení odesíláme až po opakované detekci hrozící kolize, je potřeba uvažovat delší reakční dobu. Tedy detekovat kolize do větší vzdálenosti. Kolizní vzdálenost byla nastavena na 30 cm.

5.3 Grafické uživatelské rozhraní

GUI aplikace je implemetováno pomocí grafického nástroje wxformbuilder [4]. Zdrojové soubory pro wxformbuilder naleznete ve složce *wxfb-res*. Formbuilder generuje C++ zdrojové kódy do souborů *ui.h* a *ui.cpp*. V souboru *ui_spec.h* pak naleznete specifika jednotlivých tříd generovaných formbuilderem, pro použití v naší aplikaci. Hlavní funkce aplikace jsou implementovány v souboru *main.cc*, kde naleznete třídy `MyFrame` a `MyApp`, které implementují hlavní okno aplikace.

Grafické rozhraní aplikace umožňuje:

- Výběr mapy, spouštění a zastavování lokalizace
- Volbu cíle, kam se má robot dostat
- Konfiguraci různých proměnných MCL, DBSCAN, D* a dalších použitých algoritmů
- Zobrazovat průběh lokalizace

5.4 Testování implementace

V této sekci si popíšeme jak byla aplikace testována.

5.4.1 Virtuální robot

Během první fáze vývoje byl testován především samotný modul MCL, ale i další moduly, které pro svůj běh bezpodmínečně nepotřebují skutečného robota. K tomu byl vytvořen virtuální robot, tedy robot jehož chování bylo simulováno samotným programem.

Modul MCL

Testování modulu MCL bylo prováděno tak, že byly vytvořeny simulační moduly senzorů a odometrie pro MCL. Modul odometrie kopíroval požadovaný pohyb na provedený pohyb. Modul senzorů využíval simulační model senzorů (viz. implementace - simulace senzorů) k emulaci skutečných senzorů. Také byla přidána "skutečná"pozice virtuálního robota.

Pro tento model bylo vytvořeno několik map neexistujícího prostředí (najdete je ve složce *maps*).

Během testování na tomto modelu bylo navrženo několik optimalizací MCL algoritmu (viz. optimalizace MCL)

Ostatní komponenty

Testování ostatních částí programu, kromě modulu pro komunikaci s robotem, mohlo být také provedeno s virtuálním robotem. Jednu věc však uvedený simulační model prověřit nedokáže, a to chování MCL algoritmu i algoritmu D* v dynamickém prostředí.

5.4.2 Skutečný robot

Zde si popíšeme postup při zprovoznění a testování aplikace na skutečném robotovi. Robot byl testován v areálu fakulty, v budově S.

Sériová komunikace

Prvním krokem bylo zprovoznění komunikace s robotem. Toto se povedlo vcelku dobře, až na jeden problém. A to, že mezi pracemi [7] a [9], bylo změněno nastavení sériové komunikace. Nastavení tedy je: rychlost 57600 Bps, 8 bitů dat, lichá parita a jeden stop bit.

Informace posílané robotem

Po správném nastavení se objevil další rozdíl mezi zmíněnými pracemi. A sice rozdíl ve formátu posílaných údajů, který byl upraven tak, aby obsahoval méně znaků.

Řízení robota

Následovalo ověření chování robota při příjmu řídících znaků. V tomto robot nereagoval. Při nahlédnutí do zdrojových kódů pro MCU ve FITkitu robota bylo zjištěno, že řízení ze sériového portu je zakomentováno. Po jeho odkomentování a nahrání upraveného software do MCU FITkitu, již řízení fungovalo, jak má. Výsledkem práce je tedy i takto velice mírně upravený program pro robota.

Implementace komunikačního vlákna

Předchozí kroky byly prováděny pomocí terminálu, následovalo implementování vlastního programu pro komunikaci s robotem. V našem případě se jedná o samostatné vlákno. Během této fáze nebylo příliš vhodné zkoušet robota řídit přímo na zemi. Robot byl tedy podložen tak, aby se jeho kola nedotýkala povrchu a pouze se sledovali reakce. Ovšem toho nebylo možné využít při testování určení otáčení pomocí kompasu.

V průběhu testování byl navržen způsob detekce kolizí na nižší úrovni, než prostřednictvím řídícího modulu.

Komunikační vlákno robota řídí, ale není tím, kdo říká, jak se má robot pohnout. To dělá výpočetní vlákno. Výpočetní vlákno řekne jak se má robot pohnout a komunikační vlákno zajistí vykonání pohybu.

Komunikační vlákno také výpočetnímu vláknu předává informace ze senzorů a odometrie. Ovšem komunikační vlákno je první, kdo na tyto informace může reagovat. Informací z odometrie a kompasu je využito pro samotnou kontrolu vykonávání požadovaného pohybu. Informací ze senzorů je využito pro detekci kolizí a případnému předčasnému ukončení vykonávání požadovaného pohybu.

Napětí na kontaktech baterií robota by nemělo klesnou pod 6,5V. Během testování komunikačního vlákna však bylo zjištěno několik problému, jež byly přisouzeny vlivu slábnoucích baterií a to někdy i při napětí nad 6,5V.

- Chyby odometrických měření - relativně brzy začnou informace z odometrie udávat větší ujetou vzálenost než odpovídá skutečnosti.
- Blokace kol - nedostatečné napětí pravděpodobně také způsobuje, že se kola zablokují. Komunikační vlákno pak uvízne ve vykonávání požadovaného pohybu, který se díky neotáčejícím se kolům nedokončí. Někdy stačí robota popostrčit.
- Selhání sonarů - někdy se stane, že všechny sonary posílají minimální hodnotu a to ne občas (možné chyby jsou přípustné) ale neustále. To pak vede k tomu, že se robot zlokalizuje (chybně) v nejstísněnějším místě na mapě.
- Zastavení programu robota - v případě že se robotovi nepovede inicializovat některý senzor, tak jeho program dále vůbec nepracuje. Tato situace je možné přisoudit i chybnému kontaktu. Kontaky senzorů mohou být docela náchylné na mechanickou manipulaci, ke které může docházet při zapínání robota, které je trochu neohrabané, protože vypínač se nachází pod horní plošinou robota.

Propojení grafické aplikace s komunikačním vláknem

Další fází bylo propojit komunikační vlákno s řídícím a lokalozačním modulem. Během propojování po jistou dobu nebyl vůbec použit robot. Na místo toho byly využity virtuální sériový port a emulátor sériových dat. Propojením dvou virtuálních sériových portů vznikne tzv. null modem. Ten umožňuje připojit jednu aplikaci na jeden port. Druhou aplikaci na druhý port. A obě aplikace pak spolu komunikují tak, že co jedna pošle na jeden port se objeví druhé aplikaci na jejím portu (a naopak). Na jeden port jsme tedy připojili naši aplikaci. Na druhý port byl připojen emulátor dat, a jako data byl opakovaně posílan úsek dat, jaká by mohl posílat robot.

Testování Monte Carlo lokalizace

K testování jsem si nakreslil, v běžném editoru obrázků, přibližnou mapu chodby druhého patra budovy S areálu FIT VUT. Připomeňmě, že to, že je mapa jen přibližná by vůbec nemělo vadit z principu MCL algortimu a použitého modelu dálkových snímačů.

Robotovi se povedlo úspěšně zlokalizovat a podle nalezené cesty dorazit do blízkosti požadované pozice, kde zastavil, a tato skutečnost byla oznámena v okně aplikace. Pokusů bylo provedeno několik, nicméně jen v tomto jednom prostředí. Na druhou stranu z testování s virtuálním robotem víme, že lokalizace funguje v různých prostředích (kromě těch kde jednoznačná lokalizace není možná).

V chodbě se při některých testech také pohybovali lidé, takže tím byla ověřena i funkčnost v dynamickém prostředí.

Nicméně vzhledem ke složitosti problému, není součástí mechanizmus jež je umožněn použít společně s algortimem D*. Tento mechanizmus umožňuje v dynamickém prostředí přeplánovat cestu pokud se v ní objeví překážka, o které se nevědělo. Robot jen zastaví a počká až bude objekt odtraněn (v případě člověka až ustoupí).

Pokud jde o detekci a zotavení z chyb lokalizace, tak tímto problémem jsem se podrobnější nezabýval. Nicméně protože je algortimus rozšířen o snižování počtu vzorků v případě, že je nalezeno dostatečně malé množství shluků vzorků, je zde mechanizmus, který při opětovném navýšení počtu shluků zajistí vygenerování dalších vzorků, aby množinu vzorků rozšířily. Při špatné lokalizaci se daly vysledovat jisté odchylky v celkovém součtu vah všech vzorků před normalizací (úpravou, aby tento součet byl 1). Nicméně jejich studiu nebyla věnována pozornost.

Kapitola 6

Ovládání aplikace

V této kapitole si popíšeme jak se aplikace ovládá. Veškeré ovládání probíhá prostřednictvím grafického rozhraní.

6.1 Počáteční nastavní a spouštění lokalizace

Po spuštění programu se otevře hlavní okno aplikace. Postup pro spuštění lokalizace zahrnuje následující kroky:

- Volba portu pro komunikaci s robotem - pokud nechcete použít výchozí port (COM4), je nutné kliknout na ikonku nastavení ovládání a odometrie, kde je možné port změnit. Změnu je však třeba provést před načtením mapy, jinak na tuto změnu nebude brán zřetel.
- Výběr mapy - v panelu nastavení lokalizace kliknutím na tlačítko "mapa"otevřete dialog pro výběr souboru s mapou. Mapa musí být orientována správně na sever.
- Volba cílové pozice - kliknutím pravým tlačítkem na plochu načtené mapy můžete zvolit cílovou pozici, lze ji kdykoliv za běhu lokalizace přemístit.
- Případná další nastavení - další nastavení zahrnuje nastavení DBSCAN algoritmu a redukce počtu vzorků (na panelu lokalizace). Na panelu sensorů najdete nastavení modelu Range Finders, při najetí myši na název proměnné se zobrazí stručný popis jejího významu. Na panelu odometrie, lze ještě nastavit míru šumu vneseného do MCL algoritmu při posouvání vzorků.
- Poté je ještě potřeba inicializovat/restartovat MCL modul - kliknutím na tlačítko "Restartovat MCL"se restartuje modul MCL a tím se připraví k zahájení lokalizace. Pokud nebudete volit cílovou pozici ani žádná dodatečná nastavení, je vhodné po načtení mapy okamžik počkat před restartováním modulu MCL, než se povede navázat spojení s robotem, jinak může dojít k chybě spojení (dobrým indikátorem, že můžeme restartovat je, že se na displayi FITkitu objeví nápis STOP).
- Potom už můžeme lokalizaci spustit - kliknutím na tlačítko "play"(zelený trojúhelníček na nástrojové liště)

Lokalizaci je možné kdykoliv zastavit stisknutím stejného tlačítka jako při spuštění. Přímo za běhu je možné měnit různá nastavení, některé změny však způsobí zastavení lokalizace, některé dokonce restartování MCL algoritmu (v odůvodněných případech).

6.2 Průběh lokalizace

Po spuštění lokalizace je možno na levé straně okna aplikace sledovat průběh lokalizace. Průběh lokalizace zobrazuje mapu překážek. V ní jsou zobrazeny (pomocí malých teček) všechny aktuální vzorky MCL algortimu. Pokud je platný odhad (ne nutně správný) je rovněž zobrazen, společně s kruhovými výsečemi reprezentujícími jednotlivé sonary. Rovněž pokud je pozice platná, je zobrazena i naplánovaná cesta (pokud už je spočítána).

Kapitola 7

Shrnutí

V průběhu práce byla vytvořena aplikace napsaná v C++ řídící robota vyvíjeného na UITS pomocí Monte Carlo lokalizace a algoritmu D* pro hledání cesty. Tyto algoritmy jsou krokem kupředu ve snaze vytvořit samostatně jednajícího robota.

Bylo ověřeno, že Monte Carlo lokalizace funguje i v dynamickém prostředí a že je dostatečně efektivní pro lokalizaci mobilního robota ve známém prostředí. Nicméně jsem se zabýval pouze malými uzavřenými prostory s rovným povrchem, které je snadné reprezentovat ve 2D. Takže fungování lokalizace v náročnějším prostředí nebylo prozkoumáno. Tato oblast by jistě v budoucnu zasloužila pozornost.

Aplikace vytvořená v rámci projektu, však má sloužit převážně k demonstraci tohoto algoritmu. Aby se implementace tohoto algoritmu stala skutečným přínosem pro samostatnost mobilního robota je vhodné ho implementovat do robota samotného.

Algoritmus D* nabízí způsob, jak se vyhýbat neočekávaným objektům při procházení naplánované cesty. Toto je ovšem nad rámec práce. Nicméně implementace tohoto rozšíření by byla velice přínosná pro další posun v samostatnosti mobilního robota.

Dalším námětem budoucí práce může být zdokonalení odometrie školního robota, která, jak se ukázalo, je dosti nedokonalá.

Ačkoliv je velká část aplikace napsána multiplatformně s pomocí knihovny wxWidgets, sériová komunikace je napsána pomocí WinAPI. Důvodem je, že wxWidgets nemá prostředky pro práci se sériovým portem a že neexistují ovladače Bluetooth adaptéru pro jiný operační systém než MS Windows. Mnoho lidí, stejně jako já, by jistě uvítalo možnost komunikovat s robotem i v jiném operačním systému. Momentálně to ovšem není možné, proto by se implementace ovladačů pro linuxový systém mohla rovněž stát zajímavým a perspektivním projektem.

Literatura

[1] http://merlin.fit.vutbr.cz/FITkit/.

[2] http://en.wikipedia.org/wiki/DBSCAN.

[3] http://www.wxwidgets.org/.

[4] http://www.wxformbuilder.org/.

[5] Bresenham, J. E.: Algorithm for computer control of a digital plotter. *IBM Systems Journal*, ročník 4, č. 1, leden 1965: s. 25–30.

[6] Ester, M.; Kriegel, H.; S, J.; aj.: A density-based algorithm for discovering clusters in large spatial databases with noise. AAAI Press, 1996, s. 226–231.

[7] Novotný, T.: *Řízení robota pomocí FITkitu, bakalářská práce, Brno*. FIT VUT v Brně, 2008.

[8] Peringer, P.: *Modelování a simulace, IMS, studijní opora, Brno*. FIT VUT v Brně, 2008.

[9] Sojka, S.: *Implementace robotického vysavače, bakalářská práce, Brno*. FIT VUT v Brně, 2011.

[10] Stentz, A.; Mellon, I. C.: Optimal and Efficient Path Planning for Unknown and Dynamic Environments. *International Journal of Robotics and Automation*, ročník 10, 1993: s. 89–100.

[11] Thrun, S.: *Probabilistic Robotics*. MIT Press, 2005, ISBN 0262201623.

[12] Wagner, R. J.: C++ implementation of Mersenne-Twister generator. http://www-personal.umich.edu/~wagnerr/MersenneTwister.h.

Příloha A

Obsah CD

Na přiloženém CD najdete zdrojové kódy aplikace zabalené v archivu *pc_app.zip*. Tento archiv obsahuje, kromě samotných zdrojových kódů také projekt pro prostředí wxDevC++, ve kterém byla aplikace vyvíjena, a projekt pro wxFormBuilder, pomocí kterého je vytvořeno GUI aplikace. Dále CD obsahuje zdrojové kódy pro školního robota (archiv *robot.zip*), které byly převzaty z [9] a mírně poupraveny. Na CD najdete také tento text v pdf.