VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

PLÁNOVÁNÍ CESTY PRO FORMACE ROBOTŮ

PATH PLANNING FOR ROBOT FORMATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ZDENKO HORNÁČEK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAROSLAV ROZMAN, Ph.D.**
SUPERVISOR

BRNO 2017

Abstrakt

Robotika zažíva v posledných rokoch veľký rozmach. Tento rozvoj je spôsobený neustálym technickým pokrokom, ktorého snahou je vyvíjať stále dokonalejšie roboty a z nich ďalej vytvárať spolupracujúce skupiny. Tieto spolupracujúce skupiny (formácie robotov) majú množstvo oblastí využitia od mapovania prostredia až po záchranu osôb. Táto bakalárska práca sa zaoberá návrhom algoritmu pre plánovanie cesty práve pre formáciu robotov. Využíva pri tom už existujúci plánovací algoritmus pre jedného robota - algoritmus Dijkstra. Na základe cesty vypočítanej týmto algoritmom vypočíta navrhnutý algoritmus cestu pre každého robota formácie tak, aby sa udržal tvar formácie počas jej presunu k cielu.

Abstract

Robotics has been experiencing great development in recent years. This development is due to constant technical progress and the effort to develop increasingly robust robots and to further develop co-operating groups of robots. These co-operating groups (robot formations) have many areas of use ranging from environmental mapping to search and rescue missions. This bachelor thesis deals with the design of the path planning algorithm for robot formations. It uses the already existing planning algorithm for single robot - the Dijkstra algorithm. Based on the path computed by this algorithm, the proposed algorithm will calculate the path for each robot so as to maintain the shape of the formation during its move to the goal.

Kľúčové slová

robotika, ROS, simulace, plánování cesty, formace

Keywords

robotics, ROS, simulation, path planning, formation

Citácia

HORNÁČEK, Zdenko. *Plánování cesty pro formace robotů*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Rozman Jaroslav.

Plánování cesty pro formace robotů

Prehlásenie

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Rozmana, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Zdenko Hornáček
17. mája 2017

Poďakovanie

Týmto by som chcel poďakovať svojmu vedúcemu bakalárskej práce, Ing. Jaroslavovi Rozmanovi, Ph.D., za pomoc a čas, ktorý mi venoval.

Obsah

Kapitola 1

Úvod

Robotika je stále pomerne mladé vedné odvetvie, ktoré zažíva v posledných rokoch veľký rozmach. Tento rozvoj je spôsobený veľkým technickým a technologickým pokrokom súvisiacim predovšetkým s miniaturizáciou elektroniky a stále dokonalejšími a výkonnejšími počítačmi.

Robotika sa vo všeobecnosti rozdeluje podla viacerých kritérií, ale asi najdôležitejšie delenie je na priemyslovú a experimentálnu robotiku. Do kategórie priemyselných robotov patria vo všeobecnosti univerzálne roboty, ktoré pomocou robotických ramien manipulujú s predmetmi. Túto činnosť vykonávajú automaticky podla predom zadaného programu. Experimentálna, alebo tiež kognitívna robotika sa odlišuje od priemyselnej robotiky tým, že robotické systémy sú vybavené určitým stupňom inteligencie. Do tejto kategórie patria predovšetkým mobilné roboty, robotické vozidlá a napr. robotické vysávače. Dnes sú priemyselné roboty natolko rozšírené, že je už veľká časť priemyslu plne automatizovaná. Na druhej strane stojí experimentálna robotika, ktorá ponúka široké pole uplatnenia, ale nie je ani z časti tak rozšírená. Tento stav je spôsobený najmä tým, že priemyselné roboty nepotrebujú zložitú logiku, pretože väčšinou vykonávajú len jednoduché úlohy typu uchop túto súčiastku a presuň ju na toto miesto. Taktiež veľkou výhodou priemyselných robotov oproti ľuďom je to, že sú rýchle, presné a na rozdiel od ľudí nepotrebujú odpočinok.

Mobilné roboty sú spravidla zložité zariadenia určené na plnenie komplexných úloh. Pre riešenie týchto zložitých úloh je potrebné aby tento robot disponoval ničím, čo sa dá nazvať vlastnou inteligenciou. Táto inteligencia musí byť na takej úrovni, aby robotovi umožnila bezpečne sa pohybovať po svojom okolí. Vytvoriť takúto inteligenciu nie je príliš ťažké v prípade, že sa robot pohybuje v uzavretom statickom prostredí. Problém nastane, keď sa robot dostane do reálneho sveta, kde narazí na množstvo problémov. Takým problémom môže byť napr. dynamicky sa meniace prostredie, v ktorom musí byť schopný nielen orientácie, ale musí i splniť zadanú úlohu. Problém vytvoriť takúto úroveň inteligencie je jeden z hlavných dôvodov, prečo nie sú mobilné roboty viac rozšírené.

S rozvojom robotiky prichádzajú neustále nové výzvy. Jednou takou výzvou je i ovládanie skupiny robotov tak, aby spolu fungovali ako celok. Motiváciou pre prijatie tejto výzvy je široké pole využitia spolupracujúcej skupiny robotov, či už ide o lokalizáciu či mapovanie (autonómne lietadlá - drony), vyhladávanie a záchrana osôb či sledovanie.

1.1 Stanovenie cieľa

Táto bakalárska práca sa zaoberá plánovaním cesty pre formáciu mobilných robotov. Hlavným problémom, ktorý práca rieši, je problém zostavenia a udržania formácie robotov. Formácia je skupina robotov, ktorá si udržuje určitý geometrický tvar tým, že každý robot patriaci do tejto skupiny si udržuje sadu geometrických obmedzení voči okolitým robotom. Cieľom tejto práce je analyzovať existujúce riešenia tvorby formácií robotov a vytvoriť systém, ktorý zo skupiny robotov vytvorí formáciu, ktorú udržuje počas celého presunu po mape. V tejto kapitole sa nachádza úvod do problematiky a motivácia pre štúdium tejto problematiky.

1.2 Motivácia

Okrem samotnej výzvy ovládania skupiny robotov existuje viacero výhod spojených s koordináciou formácie robotov. V prírode sa často stretávame s tým, že zvieratá sa pohybujú v skupinách, napr. kŕdle vtákov, či húfy rýb. Dôvodov pre toto správanie je niekoľko. Keď skupina zvierat skombinuje svoje zmyslové schopnosti, majú väčšiu šancu vyhnúť sa predátorom [25]. Ďalším dôvodom je ochrana mláďat pred predátormi tým, že ostanú v strede formácie [31]. Naopak predátory využívajú formácie pri lovení veľkej koristi [19]. Štúdie ukázali, že husi lietajú vo formáciach tvaru písmena „V“ preto, lebo letom cez vzduchové víry tvorené predchádzajúcou husou ušetria energiu a môžu tak doletieť ďalej [7].

1.3 Oblasti využitia formácie robotov

Formácie robotov získavajú s rozvojom vedy a samotnej robotiky stále nové využitia. V súčasnosti sa využívajú na plnenie komplexných úloh, ako je lokalizácia a mapovanie [14], [13], manipulácia s objektami a ich transportácia [30], vyhľadávanie a záchrana osôb [29] či sledovanie a monitorovanie [28]. Umelé družice vo vesmíre sú jedným z príkladov použitia formácie robotov k prieskumu a mapovaniu priestoru. K prieskumu sú formácie robotov využívané i vo vojenskom priemysle.

Kapitola 2

Základné informácie

2.1 Definícia modelu robota

V práci uvažujeme dvojkolesového robota, ktorý sa pohybuje v globálnom kartézianskom súradnicovom systéme. Stav každého robota i je reprezentovaný nasledovne:

$$q_i = [x_i, y_i, \theta_i]^T \tag{2.1}$$

$i = 1, \ldots, n$ je index robota, x_i a y_i sú súradnice robota s ohľadom na globálny súradnicový systém a θ_i udáva orientáciu robota i.

$$x_i = v_i \cos\theta_i, \quad y_i = v_i \sin\theta_i, \qquad \theta_i = \omega_i \tag{2.2}$$

$$x\sin(\theta) - y\cos(\theta) = 0 \tag{2.3}$$

Kinematický model reprezentujúci robota i je uvedený v (2.2), kde v_i a ω_i označujú lineárnu, resp. uhlovú rýchlosť vzhľadom na ťažisko robota i. Tento model má pridružené neholonomické obmedzenie (2.3), ktoré nepovoluje posúvanie robota do strán [6].

Každý robot podlieha týmto obmedzeniam hranice rýchlosti a akcelerácie:

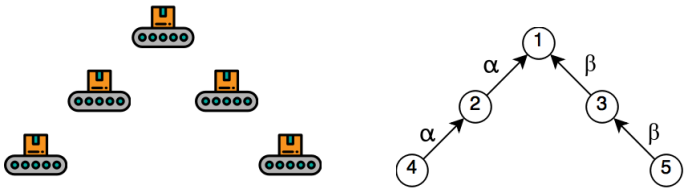
$$0 \leq v_i \leq V_{\max} \tag{2.4}$$

$$|a_i| \leq a_{\max} \tag{2.5}$$

kde $V_{\max}$ a $a_{\max}$ sú maximálne hranice lineárnej rýchlosti, resp. akcelerácie [17].

2.2 Formácia robotov

Definujeme formáciu ako orientovaný graf $G = (V, E)$, kde vrcholy V reprezentujú jednotlivé roboty a označené hrany E reprezentujú geometrické obmedzenia, ktoré sú robotmi udržované. Príklad na obr. 2.1 ukazuje formáciu robotov a grafovú reprezentáciu tejto formácie. Obmedzenia uvedené pri každej hrane sú definované ako vzdialenosť dvoch robotov a ich uhol z pohľadu robota-následovníka smerom ku svojmu predchodcovi. Pri určovaní veľkosti tohto uhla sa predpokladá, že všetky roboty majú zosynchronizovaný smer natočenia [20]. Na obr. 2.1 a v ostatku práce sú roboty natočené smerom k hornej časti stránky.



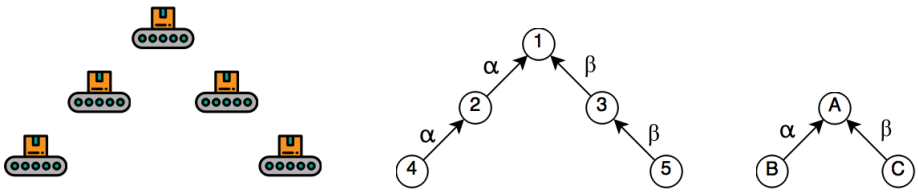
Obr. 2.1: Skupina robotov vo formácii a grafová reprezentácia tejto formácie.

2.3 Čata

Pre každú formáciu existuje aspoň jeden podgraf s nasledujúcimi vlastnosťami:

- Všetky vrcholy sú spojené jednou necyklickou cestou.
- Všetky hrany na tejto ceste majú rovnaké obmedzenia.
- Všetky vrcholy, okrem počiatočného vrcholu cesty, majú len jednu vstupnú hranu.
- Všetky vrcholy, okrem koncového vrcholu cesty, majú len jednu výstupnú hranu.

Podgraf, ktorý spĺňuje tieto obmedzenia sa nazýva *čata*. Špeciálnym prípadom čaty o veľkosti jeden je akýkoľvek samotný robot. Vzhľadom k tomu, že orientovaná cesta je vlastne usporiadaná množina vrcholov, existuje počiatočný vrchol v ceste i vrchol koncový. Počiatočný vrchol v ceste sa označuje ako *chvost* formácie a koncový vrchol je nazývaný ako *vodca* alebo *veliteľ* [21]. V tejto práci využívame pre označenie konečného vrcholu pojem *vodca*. Na obr. 2.2 sú zobrazené dva spôsoby grafickej reprezentácie formácie robotov. Prvý spôsob (na obrázku v strede) popisuje každý robot zvlášť. Druhý spôsob (na obrázku v pravo) pracuje s čatami. Teda roboty 2 a 4 sú nahradené čatou B, pretože majú rovnaké obmedzenie α a roboty 3 a 5 sú nahradené čatou C, pretože majú rovnaké obmedzenie β.



Obr. 2.2: Redukcia grafu formácie na graf čiat. Roboty 2 a 4 utvárajú čatu B a roboty 3 a 5 utvárajú čatu C.

2.4 Typické tvary fromácií

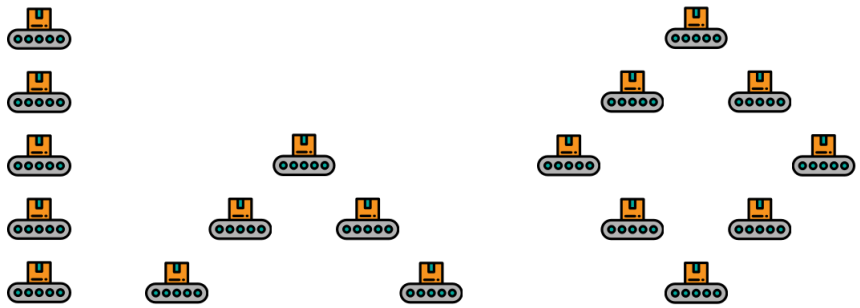
Existuje nekonečné množstvo možných formácií robotov. Avšak v literatúre sa často vyskytuje len niekoľko známych typov formácií. Sú to rad, klin, diamant a pravidelný polygón. Príklady týchto typov formácií sú zobrazené na obrázkoch 2.3 a 2.4.

rad V tomto type formácie udržuje každý robot, s výnimkou vodcu formácie, rovnaké obmedzenie so svojim najbližším susedným robotom.

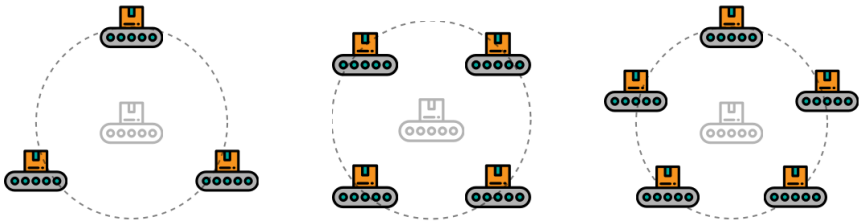
klin Tento typ formácie pozostáva z vodcu formácie a dvoch rovnako veľkých čát. Uhly, ktoré sú súčasťou obmedzenia susedných robotov v čatách, sú si v jednotlivých čatách navzájom inverzné.

diamant Tento typ formácie pozostáva z vodcu formácie, štyroch čát a jedného robota v pozícii chvosta formácie. Usporiadanie prvých dvoch čát je rovnaké ako pri formácii typu klin. Každú z týchto dvoch čát nasleduje ďalšia čata. Každá táto nasledujúca čata má uhol, ktorý je súčasťou obmedzenia susedných robotov, iverzný oproti uhlu čaty, ktorú nasleduje. Voči každej z týchto dvoch čát si robot, v pozícii chvosta formácie, udržuje samostatné obmedzenie.

pravidelný polygón Všetky roboty sú usporiadané na kružnici, ktorej stredom je virtuálny vodca. Tieto roboty sú rozmiestnené tak, že vzdialenosť medzi každými dvoma susednými robotmi je vždy rovnaká, čím vzniká pravidelná polygonálna formácia. Konkrétny tvar formácie závisí na počte členov formácie: z troch robotov vznikne pravidelný trojuholník, zo štyroch vznikne štvorec a pod.



Obr. 2.3: Typické tvary formácií robotov, zľava: rad, klin a diamant.



Obr. 2.4: Tvary pravidelnej polygonálnej formácie skupiny robotov o počte tri, štyry, resp. päť.

2.5 Štruktúry typické pre riadenie formácie

V literatúre sa vyskytujú 3 hlavné štruktúry pre riadenie formácie robotov: vodca-následovník, virtuálne a behaviorálne štruktúry, z ktorých každá má svoje výhody a nevýhody.

Vodca-následovník (leader-follower) je štruktúra, v ktorej jeden robot je považovaný za vodcu formácie a ostatné roboty ho následujú. Nevýhodou tejto štruktúry je neexistencia spätnej väzby od následovníkov k vodcovi. Teda ak následovník zlyhá v následovaní vodcu, neexistuje žiadny mechanizmus, ktorý by garantoval udržanie formácie. Na druhej strane výhodou tejto štruktúry je jej jednoduchá implementácia a porozumenie.

Virtuálna štruktúra - všetky roboty majú pevný geometrický vzťah založený na virtuálnom bode alebo virtuálnom vodcovi. Dôležitou vlastnosťou tejto štruktúry je to, že virtuálny vodca nikdy nezlyhá. Nevýhodou je, že použitím tejto štruktúry nie je možné prekonfigurovať formáciu.

Behaviorálna štruktúra sa popri hladaní cieľa sústreďuje hlavne na problémy vyhýbania sa zrážkam s prekážkami a taktiež medzi jednotlivými robotmi. Každému robotovi je adekvátne predpísané požadované správanie.

2.6 Prístupy ku komunikácii

Vo smere komunikacie medzi členmi formácie, v literatúre sa objavujú 2 prístupy: *centralizovaný* a *decentralizovaný* prístup. V centralizovanom prístupe je riadenie každého člena založené na kontrolných signáloch prichádzajúcich z centrálneho kontroléra. Centralny kontrolér príjma stavy všetkých členov. Následne, nahliadajúc na všetkých členov ako systém, centrálny kontrolér poskytne kontrolné signály. Výhodou tohto prístupu je, že v núdzových podmienkach môže ľudský operátor prevziať kontrolu. Avšak prítomnoťou porúch v centrálnom kontroléry prestáva byť systém stabilný. V decentralizovanom prístupe robí každý člen vlastné rozhodnutia založené na informáciach o ostatných členoch získaných prostredníctvom vlastných senzorov. Teda každý člen si poskytuje vlastný riadiaci vstup bez ohladu na riadiace vstupy ostatných členov.

Kapitola 3

Plánovanie cesty

Keď má robot k dispozícii mapu prostredia, v ktorom sa nachádza, odhad svojej pozície v tomto prostredí a má zadanú cieľovú pozíciu, mal by byť schopný pohybu z počiatočnej polohy do cieľovej. Plánovanie cesty potom spočíva v nájdení postupu, ako má robot dosiahnuť cielovej pozície. Pred tým, ako začneme popisovať spôsob plánovania cesty je potrebné definovať niekoľko pojmov.

Konfigurácia robotického systému je úplná špecifikácia pozície každého bodu daného systému. Označuje sa q. *Konfiguračný priestor* (C) robotického systému je priestor všetkých možných konfigurácií daného systému. Teda konfigurácia je jednoducho bod v abstraktnom konfiguračnom priestore.[8]

Pri plánovaní sa každá situácia robota nazýva stav a označuje sa ako x. Množina všetkých možných stavov sa nazýva *stavový priestor* označovaný symbolom X. Keď aplikujeme na aktuálny stav x akciu u, tak sa aktuálny stav zmení na stav x', ktorý je daný prechodovou funkciou $x' = f(x, u)$. Akcie, ktoré môžu byť aplikované na stav x, tvoria *akčný priestor* stavu x označovaný ako $U(x)$. Všetky možné akcie všetkých stavov tvoria množinu $U = \bigcup_{x \in X} U(x)$. Množina $X_G \subset X$ obsahuje prípustné cieľové stavy. Úlohou plánovania je nájsť konečnú postupnosť akcií, ktorých aplikovaním postupne transformujeme počiatočný stav x_{init} do niektorého stavu z X_G [15].

Na základné (holonomické) plánovanie cesty sa môže pozerať ako na prehladávanie konfiguračného priestoru C, v ktorom každé $q \in C$ špecifikuje pozíciu a orientáciu jednoho alebo viacerých geometricky komplikovaných telies v 2D alebo 3D svete. Úlohou plánovania cesty je vypočítať spojitú cestu z počiatočnej konfigurácie q_{init} do cieľovej konfigurácie q_{goal} [16].

3.1 Problém plánovania pohybu formácie robotov

Plánovanie pohybu pre skupinu robotov je problém. Bezkolízna cesta z počiatočnej konfigurácie robotov do cieľovej konfigurácie robotov implikuje, že pri každom kroku nedochádza ku kolízii medzi robotom a prekážkou alebo medzi robotom a iným robotom. Riešenie tohto problému musí zvládať dve úlohy. Musí nájsť také cesty pre jednotlivé roboty, ktoré garantujú len to, že nenastanú kolízie s prekážkami. A druhou úlohou riešenia je koordinácia týchto ciest tak, že žiadne dva roboty sa nedostanú do vzájomnej kolízie. Práve táto druhá úloha robí problém plánovania pohybu formácie robotov značne ťažším, než je to v prípade jednoho robota. Sú dva klasické prístupy k riešeniu tohto problému: *centralizované* a *decentralizované plánovanie* [9].

3.1.1 Centralizované plánovanie

Centralizované plánovanie neuvažuje skupinu robotov ako jednotlivé roboty, ale pracuje s nimi akoby tvorili jednoho robota tvoreného viacerými časťami. Konfiguračný priestor C je výsledkom karteziánskeho súčinu konfiguračných priestorov všetkých robotov. Koordinácia robotov je dosiahnutá jednoduchým spôsobom: bezkolízna konfigurácia v C popisuje konfiguráciu všetkých robotov a tak zaisťuje, že žiadny robot nie je v kolízii s prekážkou alebo iným robotom [9].

3.1.2 Decentralizované plánovanie

Decentralizované plánovanie pracuje v dvoch fázach. Na začiatku sa vypočítajú bezkolízne cesty pre každého robota zvlášť. Tieto výpočty neberú do úvahy ostatné roboty, počítajú iba s prekážkami prostredia. V druhej fáze je dosiahnutá koordinácia vypočítaním relatívnych rýchlostí jednotlivých robotov na ich individuálnych cestách tak, aby medzi robotmi nedošlo ku kolízii. Algoritmus decentralizovaného plánovania je neúplný, napriek tomu, že algoritmy využité v obidvoch fázach sú úplné: môže byť nemožné skoordinovať niektoré cesty nájdené v prvej fáze plánovania tak, aby nedošlo ku kolízii medzi dvoma odlišnými robotmi [9].

Kapitola 4

ROS - Robot Operation System

Robot Operation System, ďalej len ROS, je felixibilný framework pre písanie robotického softvéru. Je to súbor nástrojov, knižníc a konvencií, ktoré majú za cieľ zjednodušiť úlohu tvorby komplexného a robustného ovládania robota v širokej škále robotických platforiem. ROS nie je klasický operačný systém, pretože pre svoj beh potrebuje iný operačný systém. Doporučeným systémom pre vývoj je Ubuntu, pre ktoré sú vydávané predkompilované balíčky. Okrem Ubuntu sú vydávané predkompilované balíčky i pre Debian.

ROS softvér môže byť rozdelený do 3 skupín:

- nástroje určené na vytváranie a distribuovanie ROS softvéru, tieto nástroje sú nezávislé na jazyku i platforme;
- implementácie hlavných klientských knižníc pre ROS;
- balíčky obsahujúce kód aplikácií, ktoré využívajú klientské knižnice pre ROS.

Prvé dve skupiny sú dostupné pod BSD licenciou, a ako také sú open source softvér a zdarma pre nekomerčné i komerčné využitie. Väčšina ostatných balíčkov je licencovaná pod rôznymi open source licenciami. Tieto balíčky implementujú bežne používané funkcie a aplikácie, ako sú hardvérové ovládače, robotické modely, dátové typy, plánovanie, vnímanie, simultánna lokalizácia a mapovanie, simulačné nástroje a ďalšie algoritmy.

Hlavné ROS klientské knižnice (C++, Python, LISP) sú orientované na Unixové systémy, a to predovšetkým kvôli ich závislosti na veľkých súboroch open source softvérových závislostí. Ubuntu Linux je uvedený ako „podoporovaný“ pre všetky tieto klientské knižnice, zatiaľčo iné varianty, ako je Fedora Linux, Mac OS X a Microsoft Windows sú označené ako „experimentálne“ a sú podporované komunitou. Tieto obmedzenia nezdiela natívna Java ROS klientská knižnica. Táto knižnica umožnila písanie softvéru založeného na ROS pre Android OS. Okrem toho umožnila integráciu ROS do oficálne podporovaného nástroja Matlab, ktorý môže byť použitý v Linuxe, Mac OS X i Microsoft Windows.

Táto kapitola čerpá z [2] a [22].

4.1 Súborový systém v ROS

Zdroje ROS sú organizované do hierarchickej štruktúry na disku. Dva dôležité koncepty vynikajú:

Balíček (package) je základná jednotka v rámci ROS softvérovej organizácie. Balíček je adresár obsahujúci uzly (uzly sú popísané nižšie), externé knižnice, dáta, konfiguračné súbory a jeden xml konfiguračný súbor nazvaný Manifest.xml.

Stoh (stack) je zbierka balíčkov. Ponúka rad funkcií, ako je navigácia, polohovanie, atď. Stoh je adresár obsahujúci adresáre balíčkov a konfiguračný súbor s názvom stack.xml.

Ďalším dôležitým pojmom je **distribúcia**, ktorá pomenúva zbeirku stohov rovnakej verzie. Najnovšia distribúcia je označovaná kódovým menom *Kinetic Kame* (obr. 4.1).

Obr. 4.1: Najnovšia distribúcia ROS je označovaná ako *Kinetic Kame*.

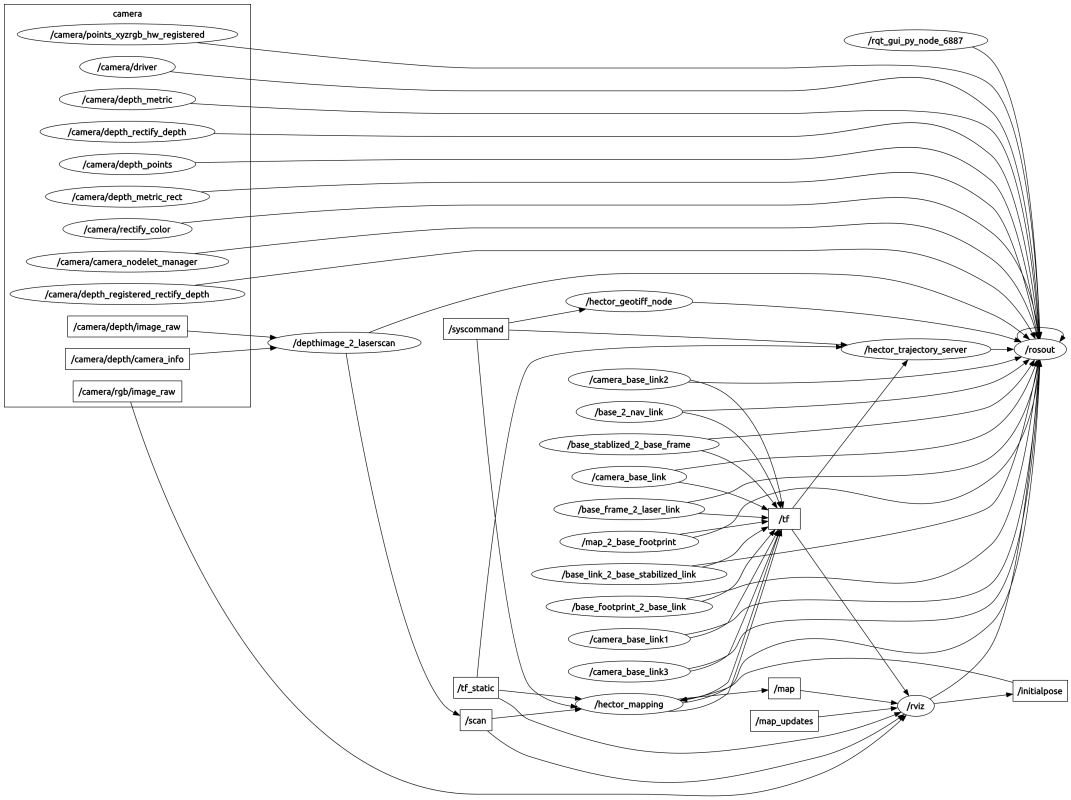
4.2 Základné pojmy

Základný princíp ROS je paralelné spustenie veľkého počtu modulov, ktoré musia byť schopné vymieňať si medzi sebou dáta synchrónne alebo asynchrónne. Okrem toho, ROS potrebuje spravovať tieto výmeny, aby mohol zaistiť efektívny prístup ku zdrojom robota.

Základným konceptom implementácie ROS sú *uzly*, *správy*, *témy* a *služby*.

Uzly (nodes) sú procesy, ktoré vykonávajú výpočty. ROS je navrhnutý ako modulárny systém pozostávajúci z mnohých uzlov. Termín *uzol* je zameniteľný s pojmom *softvérový modul*. Použitie termínu *uzol* vychádza z vizualizácie aplikácie v ROS za behu: pri veľkom počte bežiacich uzlov je vhodné zobraziť komunikáciu peer-to-peer ako

graf s procesmi tvoriacimi uzly grafu a peer-to-peer prepojenia ako spojnice týchto uzlov (obr. 4.2).



Obr. 4.2: Schéma uzlov zložitejšej aplikácie. Uzly sú zobrazené v elipsách, témy v obdĺžnikoch.

Master je centrálny prvok, ktorý zaisťuje služby pre deklaráciu a registráciu uzlov, čím umožňuje uzlom nájsť sa navzájom a vymieňať dáta. Master zahrňuje dôležitú komponentu nazvanú *Parameter Server*. Ako názov napovedá, ide o akúsi centralizovanú databázu, v ktorej uzly môžu ukladať dáta, a pritom zdieľať globálne parametre.

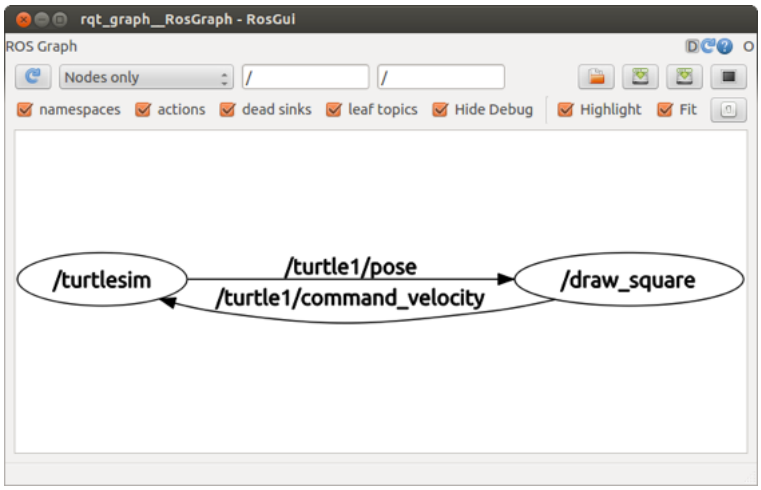
Správy (messages) . Uzly medzi sebou komunikujú zasielaním správ. Správa je striktne typovaná dátová štruktúra. Podporované sú štandardné primitívne typy (integer, string, boolean, atď.) a taktiež polia primitívnych dátových typov a konštanty. Okrem toho môže správa obsahovať iné správy a polia iných správ vnorené ľubovolne hlboko.

Dáta sa vymieňajú asynchrónne prostredníctvom *tém* a synchrónne prostredníctvom *služieb*.

Téma (topic) je spôsob prenosu dát založený na systéme *odoberania* (subscribe) a *publikovania* (publish). Jeden alebo viac uzlov môžu publikovať dáta na určitú tému a zároveň jeden alebo viac uzlov sa môžu prihlásiť na odoberanie dát z tejto témy. V istom zmysle ide o asynchrónnu zbernicu správ. Toto poňatie asynchrónnej, many-to-many zbernice má zásadný význam v distribuovanom systéme. Téma je typovaná, čo znamená, že typ dát publikovaných je vždy štruktúrovaný rovnakým spôsobom.

Služby (services) zabezpečujú synchrónnu výmenu dát medzi dvoma uzlami. Každá služba je definovaná reťazcom, ktorý ju pomenúva, a dvojicou striktne typovaných správ: jedna pre požiadavku (request) a druhá pre odpoveď (response). Toto je analogické k webovým službám, ktoré sú definované pomocou URI a majú požiadavky a odpovede určitých definovaných typov. Na rozdiel od tém, iba jeden uzol môže inzerovať (advertise) službu určitého mena.

Pomocou utility *rqt_graph*, ktorá je súčasťou ROS, je možné zobraziť aktuálne bežiace uzly. Na obr. 4.3 sú uzly označené elipsami a šípky reprezentujú spojenia cez témy. Vidíme, že uzol */turtlesim* publikuje dáta na tému */turtle1/pose*. Unikátnym odoberateľom tejty témy je uzol */draw_square*. Druhá šípka korešponduje s témou */turtle1/command_velocity*. Táto téma je použitá na posielanie dát z uzla */draw_sqaure* do */turtlesim* uzla.

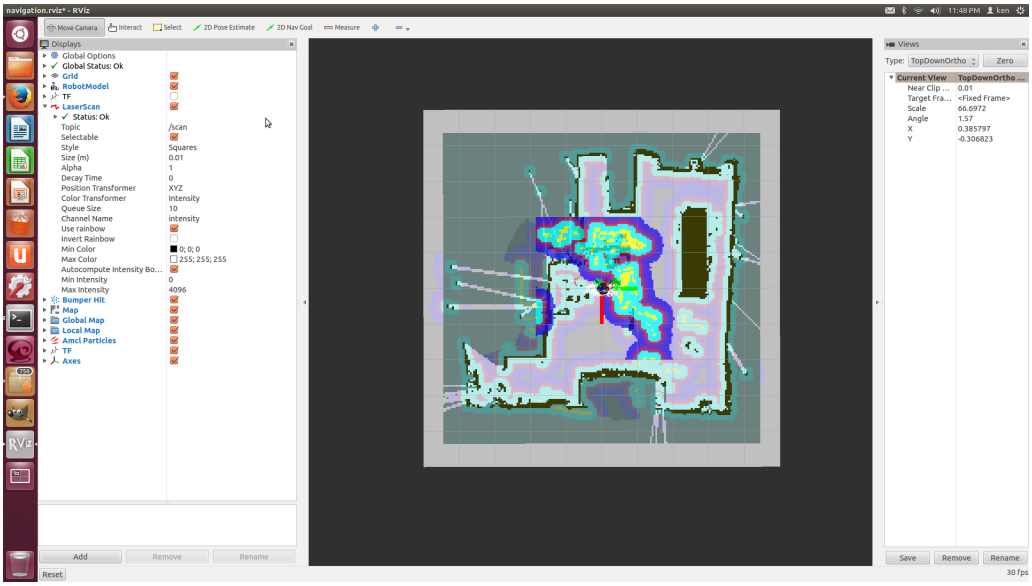


Obr. 4.3: Vizualizácia bežiacich uzlov pomocou utility rqt_graph.

4.3 Užitočné balíčky

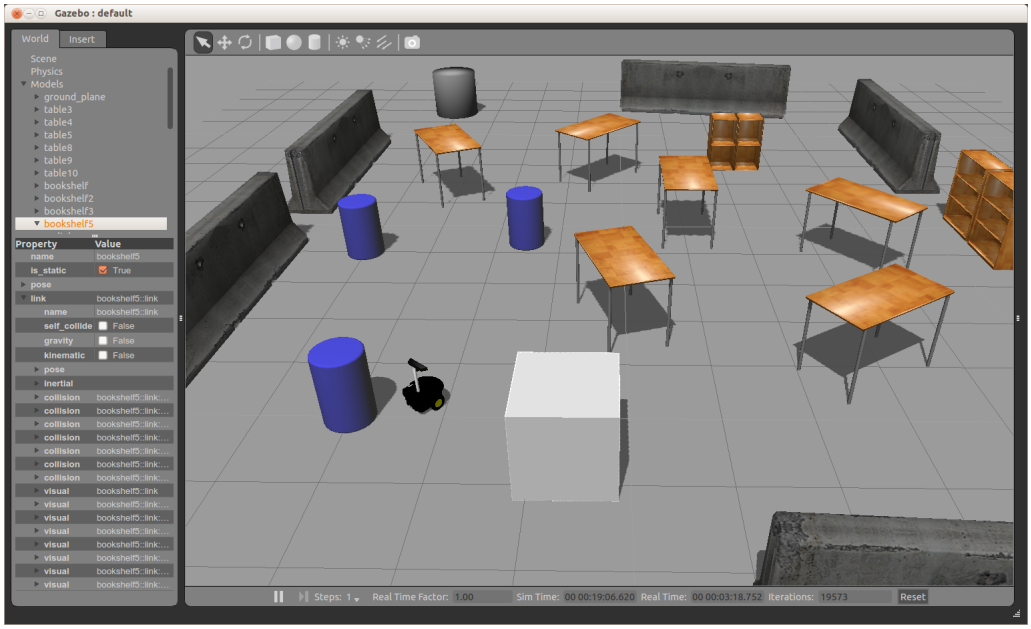
Rviz (obr. 4.4) je 3D vizualizér pre zobrazenie dát zo senzorov a stavových informácií z ROS. Použitím *rviz* je možné vizualizovať aktuálnu konfiguráciu robota na virtuálnom modele robota. Keďže nemáme k dispozícii skutočné roboty, možnosť použiť virtuálne modely robotov je pre nás dôležitá. Rviz dokáže priamo zobrazovať reprezentáciu senzorických hodnôt prichádzajúcich cez ROS témy vrátane dát z kamery, zo sonaru a iných. Ktorúkoľvek vizuálnu informáciu môžete okamžite zobraziť alebo naopak schovať. 3D vizualizáciou, ktorú rviz ponúka, je možné navigovať pomocou myši.

Gazebo (obr. 4.5) je 3D dynamický simulátor so schopnosťou presne a efektívne simulovať populácie robotov v komplexnom vnútornom i vonkajšom prostredí. Je podobný herným enginom, ale ponúka simuláciu fyziky na oveľa vyššom stupni vernosti, sadu senzorov a rozraní pre užívateľov i programy. Typické príklady použitia simulátora Gazebo zahŕňajú testovanie robotických algoritmov, navrhovanie robotov či vykonávanie regresného testovania s reálnymi scenármi. Medzi kľúčové črty Gazeba patrí



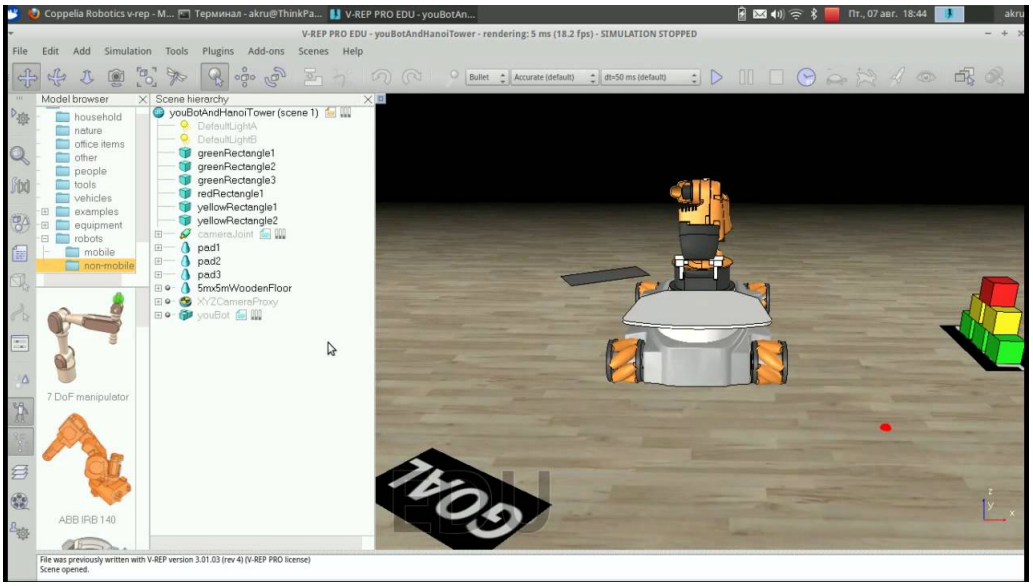
Obr. 4.4: Vizualizácia mapy prostredia prostredníctvom balíčka *rviz*. Približne v strede mapy sa nachádza model robota a farebne sú označené časti mapy, ktoré robot svojimi senzormi preskúmal. Na ľavej strane je viditeľný panel, v ktorom je možné nastaviť, ktoré informácie sa majú zobrazovať.

množstvo fyzických enginov, bohatá knižnica robotických modelov a prostredí, široká škála senzorov a priaznivé programové a grafické rozhranie [3].



Obr. 4.5: Vizualizácia robota v prostredí s prekážkami pomocou balíčka *gazebo*.

V-REP je robotický simulátor s integrovaným vývojovým prostredím. Je založený na distribuovanej riadiacej architektúre: každý objekt/model môže byť individuálne kontrolovaný pomocou ROS uzla, pluginu, vzdialeného API klienta, alebo vlastného riešenia. Vďaka tomu je V-REP veľmi univerzálny a je ideálny pre multi-robotické aplikácie [1]. V-REP nepatrí medzi klasické ROS balíčky, ale môže sa správať ako ROS uzol, s ktorým môžu iné ROS uzly komunikovať.



Obr. 4.6: Užívateľské rozhranie ROS pluginu *V-REP* .

Kapitola 5

Existujúce riešenia

Existuje viacero prác venujúcich sa danej tématike. V nasledujúcich kapitolách sú vybrané niektoré z tých zaujímavejších. Riešenia, na ktoré sme narazili, no v nasledujúcich kapitolách ich nespomíname sú [33], [12], [11], [27] a [23].

5.1 Xiao

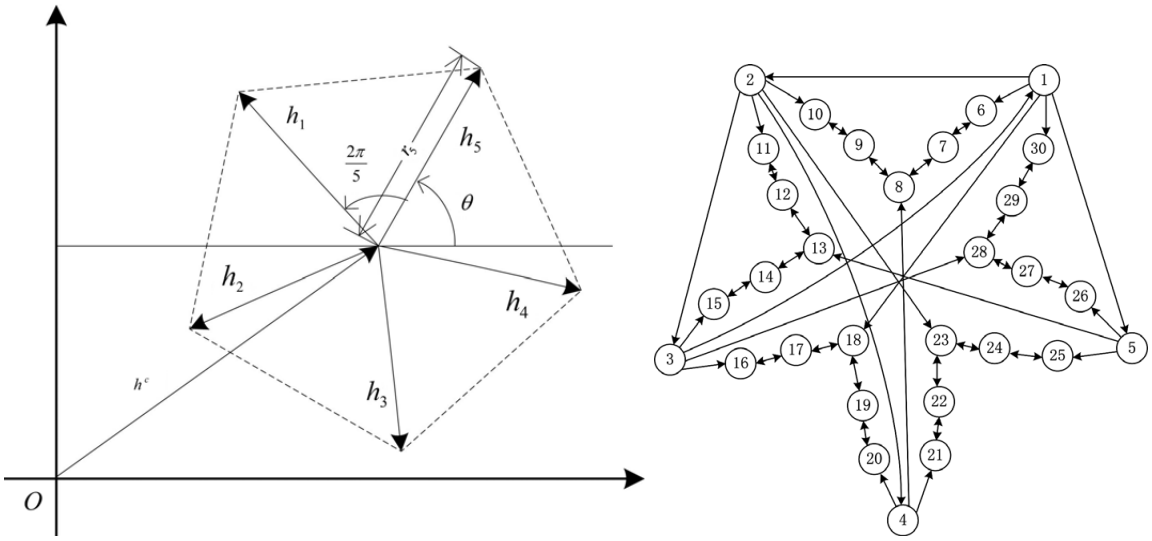
V práci [32] je popísaný kontrolný rámec pre ovládanie formácie s veľkým počtom robotov. V tomto rámci rozdelili informácie o formácii do dvoch samostatných častí: globálne informácie a lokálne informácie. Globálne informácie o formácii rozhodujú o geometrickom vzorci požadovanej formácie. Predpokladá sa, že iba malé množstvo robotov môže tieto informácie získať. Roboty, ktoré globálne informácie získajú sú zodpovedné za navigáciu celej formácie a preto sú označované pojmom vodca (leader). Tento pojdem je ale odlišný od toho používaného pri ovládaní formácie typu vodca-následovník (leader-follower), ktorý je definovaný topológiou toku informácií, na rozdiel od typov získaných informácií. Ostatné roboty regulujú svoju pozíciu distribuovaným spôsobom na základe lokálnych informácií. Tento prístup môže znatelne znížiť objem prenášaných dát a zároveň umožňuje vytvorenie rôznych komplexných formácií, najmä pri veľkom počte robotov a častých zmenách formácie.

Na obrázku 5.1 je zobrazený príklad formácie s tridsiatimi robotmi. Vľavo je zobrazený rámec tejto formácie pozostávaujúci z piatich vodcov. Skupina vodcov drží tvar pravidelného päťuholníka (predpokladá sa, že $h_i = h_{i+1}$, pre všetky $i = 1, 2, ..., 4$ a uhol medzi jednotlivými vodcami je $\frac{2\pi}{5}$). Formáciu tvorí okrem piatich vodcov i dvadsaťpäť následovníkov. Na obrázku vľavo je zobrazená topológia interakcií medzi robotmi tejto formácie.

Autori zmieňujú niektoré problémy, ktoré tento rámec nerieši, ale je ich potrebné adresovať. Je to napríklad návrh takých distribuovaných protokolov pre vodcov, ktoré dokážu bez referenčnej trajektórie viesť celú formáciu robotov cez prekážky až do cieľa či zabezpečenie vyhnutia sa kolízií medzi jednotlivými robotmi.

5.2 Ren

Predpokladá sa, že v typickej formácii typu vodca-následovník (leader-follower) existuje iba jeden vodca. Ten má ako jediný k dispozícii informácie o trajektórii cesty celej skupiny. Tieto informácie sú buď predprogramované do vodcu, alebo ich vodca získa z externého



Obr. 5.1: Na obrázku vľavo je zobrazený rámec formácie. Vpravo je zoprazená topológia toku informácií medzi robotmi tejto formácie.

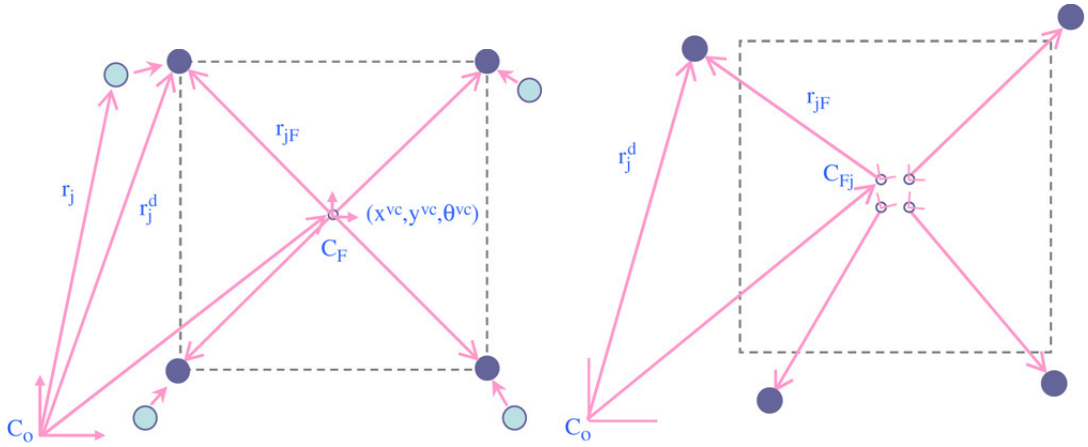
zdroja. Formácia je následne zostavená na základe reakcie ostatných členov skupiny na pohyb vodcu. Fakt, že v skupine robotov sa nachádza iba jeden vodca implikuje, že prístup k ovládaniu formácie typu vodca-následovník je jednoduchý na pochopenie i implementáciu. Zároveň to ale znamená i to, že stratou vodcu sa celá skupina rozpadne. Ďalším problémom tohto prístupu je neexistencia spätnej väzby od následovníkov k vodcovi. Dôsledkom môže byť napríklad oddelenie následovníka zo skupiny, pretože nedokáže presne sledovať pohyb vodcu.

Decentralizované alebo distribuované kooperatívne stratégie ovládania formácie prekonávajú tieto problémy. Pri týchto prístupoch robí každý robot svoje vlastné rozhodnutia na základe stavov jeho najbližších susedov. To umožňuje skupine dosiahnuť cieľ i v prípade, že dôjde k zlyhaniu niektorého z jej členov.

Decentralizované alebo distribuované kooperatívne prístupy riadenia využívajú konsenzusné algoritmy, ktoré sa zameriavajú na to, aby informačné stavy všetkých robotov mali spoločnú hodnotu. Vo formácii so statickým centrálnym bodom sa tieto algoritmy využívajú na zostavenie formácie. Požiadavkou pre zostavenie formácie je potom to, že všetky roboty sa zhodnú na centrálnom bode formácie a každý robot obsahuje informáciu o požadovanej odchýlke od daného bodu. Na udržanie formácie pri pohybe použitím týchto algoritmov je potrebné aby každý robot mal informáciu o spoločnej rýchlosti skupiny alebo o požadovanej trajektórii skupiny. Tento prístup ale nepočíta s dynamickým prostredím.

Obrázok 5.2 zobrazuje situáciu, kde má skupina štyroch robotov utvoriť formáciu na základe určitých geometrických obmedzení od virtuálneho stredu formácie. Na obrázku vľavo roboty najskôr použili konsenzusný algoritmus, aby mali všetci členovia formácie identickú informáciu o pozícii virtuálneho stredu formácie a následne túto formáciu utvorili. Obrázok vpravo zobrazuje takú situáciu, v ktorej roboty konsenzusný algoritmus nepoužili a nastala nekonzistencia v informáciach jednotlivých robotov o pozícii virtuálneho stredu formácie. To má za následok, že formácia robotov nedrží požadovaný tvar.

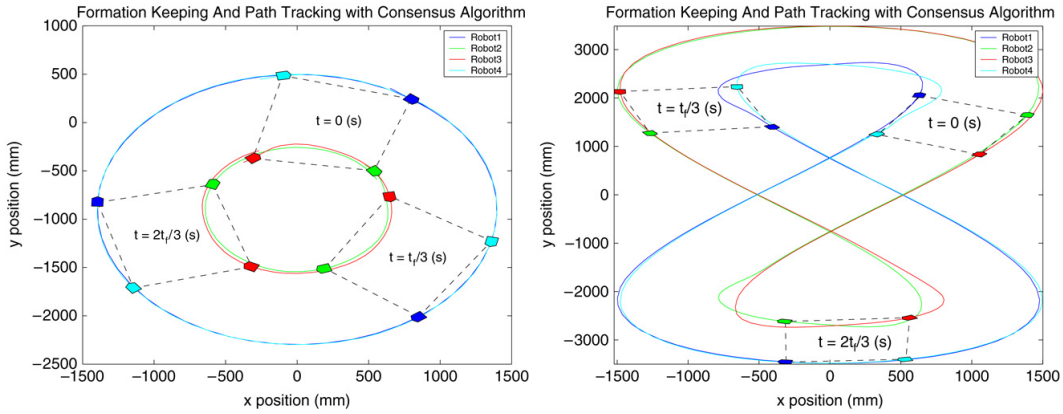
V práci [24] navrhli jednotnú, distribuovanú architektúru pre kontrolu formácie. Táto architektúra umožňuje ľubovolný počet vodcov a nijak neobmedzuje tok informácií me-



Obr. 5.2: Formácia zložená zo štyroch robotov. Vľavo je použitý konsenzusný alogritmus, vpravo nie.

dzi robotmi bez pridania zložitosti. Najmä rozšírený konsenzusný algoritmus je aplikovaný na skupinovej úrovni na odhadnutie informácie o s časom sa meniacej skupinovej trajektórii distribuovaným spôsobom. Na základe odhadnutej informácie o skupinovej trajektórii je na úrovni riadenia robota aplikovaná distribuovaná stratégia riadenia formácie. Túto architektúru experimentálne implementovali a overili.

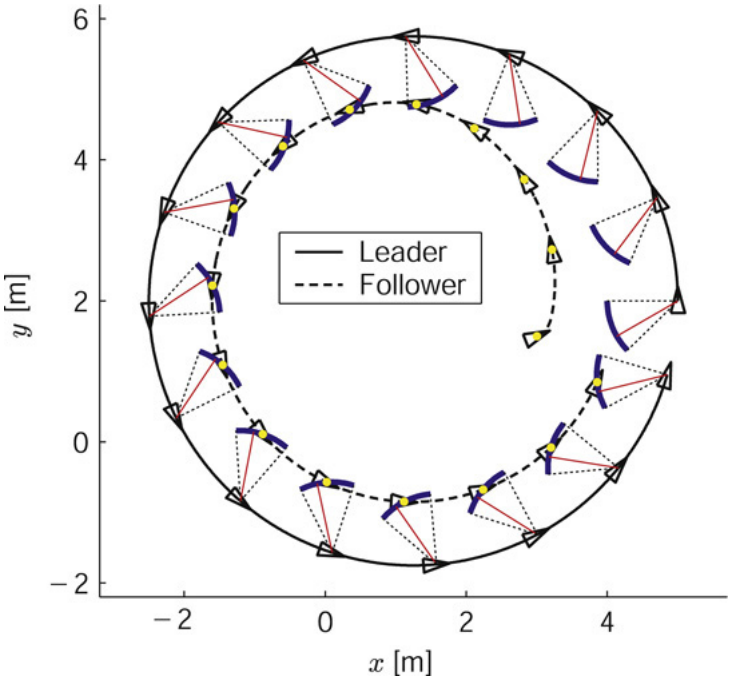
Obrázok 5.3 zobrazuje výsledok experimentov s formáciu pozostávajúcou z jedného vodcu a troch následovníkov. Na obrázku vľavo vodca následoval kružnicovú trajektóriu. Vpravo vodca následoval trajektóriu tvaru číslice 8. Z grafov je zreteľné, že roboty držali požadovaný tvar počas celého trvania cesty a odchýlka od predpísanej virtuálnej trajektórie bola minimálna.



Obr. 5.3: Formácia zložená zo štyroch robotov. Vľavo je použitý konsenzusný alogritmus, vpravo nie.

5.3 Consolini

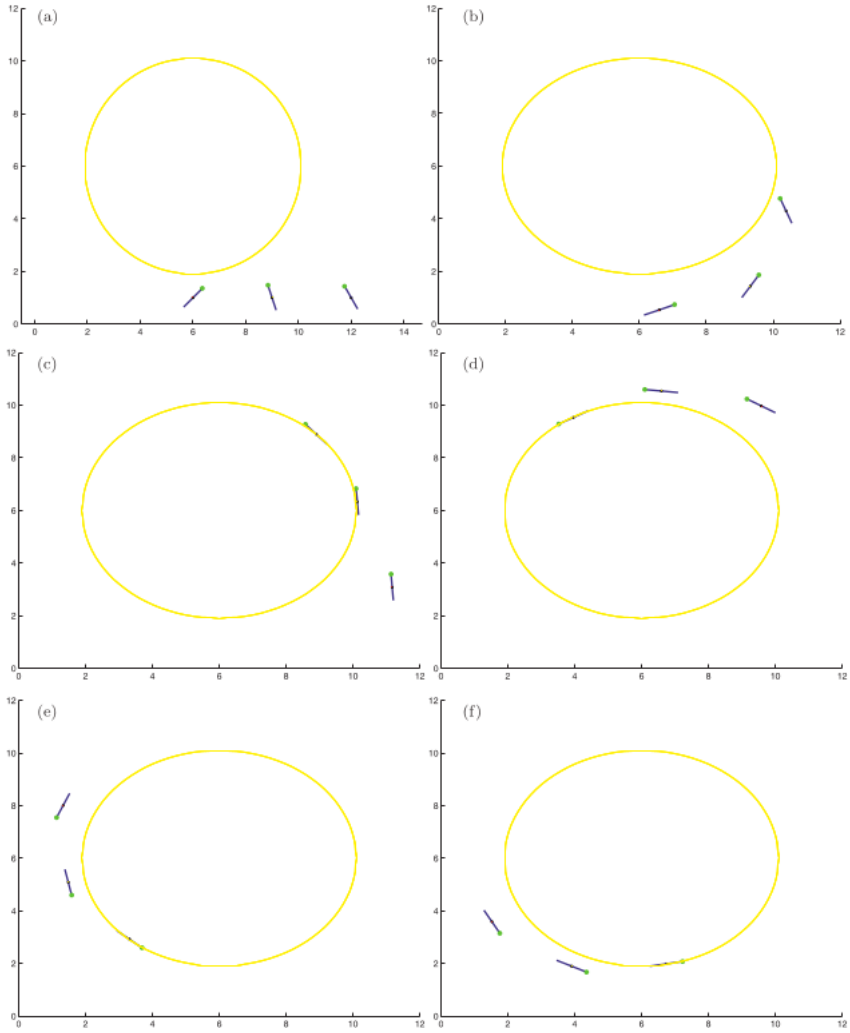
Práca [10] sa zaoberá riadením formácie neholonomných mobilných robotov s obmedzenými kontrolnými vstupmi. Riadenie je typu vodca-následovník (leader-follower), tak isto ako v predchádzajúcej kapitole. Autori prišli s alternatívnym spôsobom riešenia tohto typu riadenia. Navrhli také nastavenie, kde požadovaný uhol medzi vodcom a následovníkom bude meraný v rámci následovníka a nie v rámci vodcu ako je to u väčšiny ostatných riešení. Preukázali, že tento prístup zaručuje réžiu pri riadení a taktiež zaručuje hladšie trajektórie následovníkov než u bežných riešení. Na základe navrhnutého nastavenia našli vhodné podmienky pre vodcovskú rýchlosť a zakrivenie trajektórie také, že následovník, dodržujúc vlastné obmedzenie rýchlosti, sa dostane do formácie a udrží ju. Z geometrického hladiska nie je pozícia následovníka voči pozícii vodcu pevne zafixovaná. Naopak, pozícia následovníka sa mení, ale vždy sa nachádza na kružnicovom oblúku so stredom vo vodcovskom referenčnom rámci. Hlavný prínos práce je sformulovaný v dvoch teorémoch. Prvý teorém udáva dostatočné a potrebné podmienky pre obmedzenie vodcovskej a následovníckej rýchlosti potrebné pre existenciu riadiaceho pravidla, ktoré umožní následovníkovi udržať formáciu nezávisle od trajektórie vodcu. Druhý teorém udáva dostatočné podmienky a riadiace pravidlá pre následovníka tak, aby mohol asymptoticky dosiahnuť formáciu pri akýchkoľvek podmienkach a akomkoľvek pohybe vodca a pritom rešpektovať obmedzenia rýchlosti. Tieto dva teorémy overili experimentom, ktorého výsledok je na obrázku 5.4.



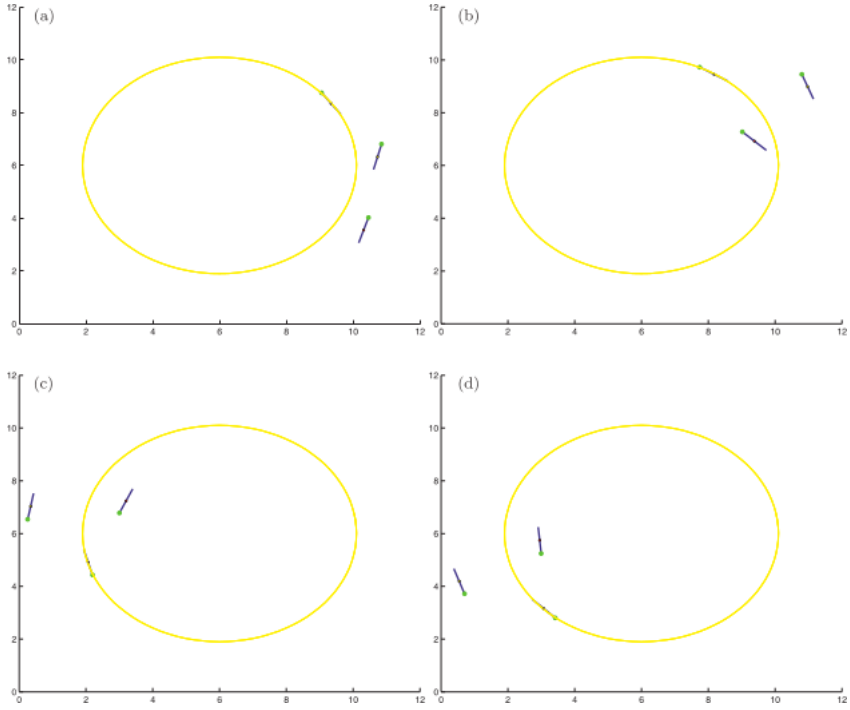
Obr. 5.4: Trajektória vodcu (zobrazená plnou čiarou) a následovníka (prerušovanou čiarou) v prípade, že sa následovník zapája do formácie a vodca nasleduje kružnicovú trajektóriu.

5.4 Mastellone

Návrhom kontroléru, ktorý garantuje koordinované následovanie a vyhýbanie sa prekážkam pre skupinu neholonomných robotov, sa zaoberá práca [18]. Autori predpokladajú, že každý robot pozná svoju pozíciu a vie detekovať prítomnosť objektu v danom rozsahu. Tento predpoklad aplikujú pri riadení formácie. Ďalej v práci adresujú problém koordinovaného následovania, kde cieľom je pohybovať skupinou robotov po generovanej trajektórii tak, aby bola po celý čas udržaná formácia. Použili decentralizovanú architektúru, v ktorej sú kontroléry implementované lokálne na každom robotovi. Časť kontroléra, ktorá je zodpovedná za následovanie, používa lokálne informácie o aktuálnej pozícii robota i vodcu či virtuálneho vodcu (stred formácie). Druhá časť kontroléru je zodpovedná za vyhýbanie sa prekážkam. Používa informácie o pozícii prekážok a iných robotov v dosahu. Konkrétnejšie, táto druhá časť pracuje v reálnom čase a používa lokálne definované potenciálové funkcie rôzneho druhu, ktoré vyžadujú iba to, aby každý robot detekoval objekty v jeho okolí. V poslednej časti práce autori vykonávajú rozsiahle experimenty (obr. 5.5 a 5.6) pre potvrdenie teoretických výsledkov.



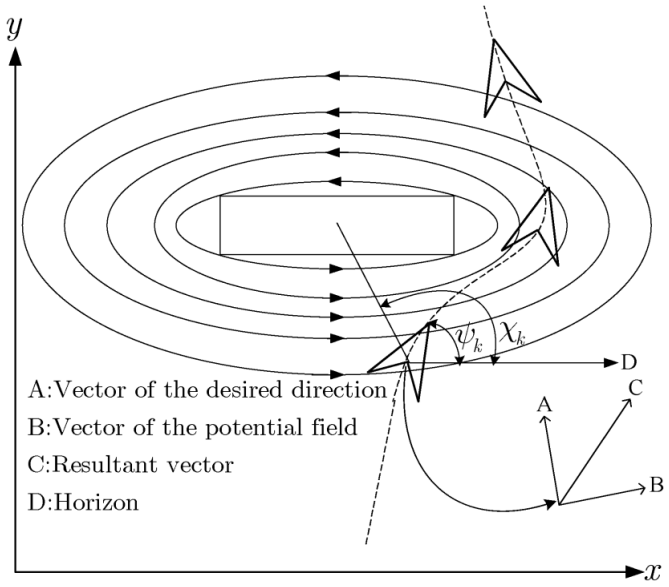
Obr. 5.5: Utvorenie formácie typu rad a jej pohyb po kružnicovej trajektórii.



Obr. 5.6: Zmena formácie z typu rad na typ klin.

5.5 Rezaee

Autori práce [26] navrhli pužitie metódy odpudivých síl pri riadení formácie skupiny robotov založenej na virtuálnej štruktúre. Táto metóda sa používa hlavne na vyhýbanie prekážkam a kolíziam medzi robotmi. Táto práca adresuje techniku pre riadenie formácie mobliných robotov založenej na decentralizovanej virtuálnej štruktúre. Mobilné roboty sú umiestnené na kružnici s preddefinovaným polomerom okolo virtuálneho bodu považovaného za stred kružnice. Pre dosiahnutie pravidelnej polygonálnej formácie sa každý robot modeluje ako elektrický náboj. Ak sú elektrické náboje identické a štruktúra robotov je podobná, dané roboty sa navzájom odpudzujú, čím vznikne stabilná formácia s rovnakými rozostupmi medzi susednými robotmi. Vo väčšine použitých prístupov využívajúcich virtuálnu štruktúru nie je možné zmeniť konfiguráciu formácie. V tejto práci ale autori navrhli prítup umožňujúci každému robotovi nájsť svoju pozíciu v polygonálnej formácii autonómne. To umožňuje pri zmene počtu robotov upraviť formáciu na inú pravidelnú polygonálnu formáciu. Problém vyhýbania sa zrážkam s prekážkami je v literatúre riešený viacerými spôsobmi. Napr. každá prekážka je obalená do konvexného polygónu a produkujú odpudivé sily pre vyhnutie sa kolízii, alebo je použité potenciálové pole na vyhnutie sa prekážkam. Tieto prístupy majú však spoločný problém. Ak sa robot pohybuje opačným smerom ako udáva potenciálové pole, môže sa stať, že robot túto prekážku nedokáže obísť a môže uviaznuť v pozícii lokálneho minima, keďže výsledný vektor pohybu robota a odpudivý vektor prekážky môže byť nulový. Autori práce vyriešili tento problém použitím rotačných odpudivých vektorov. To znamená, že v použitých rotačných potenciálových poliach (na obr. 5.7) sú smery vektorov upravené v smere prichádzajúceho mobilného robota a tým ho vedú k jeho cieľu bez možnosti uviaznutia v pozícii lokálneho minima.



Obr. 5.7: Rotačné potenciálové pole okolo prekážky a trajektória prichádzajúceho mobilného robota. A: vektor želaného smeru, B: vektor potenciálového pola, C: výsledný vektor, D: horizont.

Kapitola 6

Návrh riešenia

Obsahom tejto kapitoly je navrhnuté riešenie problému plánovania cesty pre formáciu robotov. Keďže ROS framework má v zdrojových knižniciach niekoľko plánovacích algoritmov (DWAPlanner [5], CarrotPlanner [4]) využijeme jeden z nich a sústredíme sa na riešenie problému vytvorenia formácie a udržania jej tvaru počas pohybu po ceste.

Rozhodli sme sa vytvoriť riešenie vychádzajúce z práce [24] popísanej v kapitole 5.2. Využijeme riadenie formácie typu vodca-následovník, kde pozícia každého robota bude vypočítavaná na základe stavov najbližších susedov, čím umožníme dosiahnuť skupine cieľ i v prípade zlyhania niektorého z jej členov. Vo formácii sa určí centrálny bod, ktorý bude slúžiť na zostavenie formácie - v systéme bude informácia o odchýlke od tohto bodu pre každý robot, na základe čoho bude vypočítaná požadovaná poloha robota pre zostavenie formácie. Pre udržanie formácie bude dostupná informácia o požadovanej trajektórii skupiny. Z tejto požadovanej trajektórie sa bude počítať požadovaná poloha pre každý robot opäť na základe jeho odchýlky od tejto trajektórie.

6.1 Simulácia

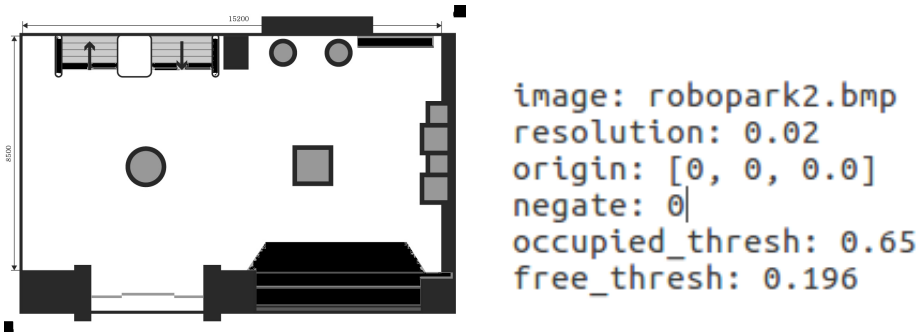
Nemáme k dispozíci dostatočný počet robotov rovnakého typu, preto využijeme simuláciu. K samotnej simulácii využijeme balíčky *stage_ros* a *rviz*. Balíček stage_ros simuluje svet definovaný v (.world) súbore, ktorý obsahuje detaily pre senzory, robotov a prekážky v simulovanom svete. Rviz je 3D vizualizačný nástroj, ktorý použijeme pre zobrazenie dát zo senzorov a stavových informácií z ROS. V simulácii je potrebné definovať následovné:

Model robota sme zvlolili robota *kobuki* s hexagonálnym tvarom podstavy a kinectom slúžiacim ako 3D senzor (obr. 6.1).

Obr. 6.1: Robot kobuki s hexagonálnym tvarom podstavy a kinectom.

Navigáciu zaistíme použitím balíčka *turtlebot navigation*.

Mapu prostredia, v ktorom sa robot pohybuje. Táto mapa sa definuje obrázkom a popisným .yaml súborom, ktorý obsahuje informácie o danej mape (obr. 6.2). Tieto informácie zahrňujú rozlíšenie mapy (udávané v metroch na pixel), počiatok mapy (2D pozícia lavého spodného pixelu v mape) a ďalšie. Na použitie mapy v simulácii je potrebné použiť balíček *map server*.



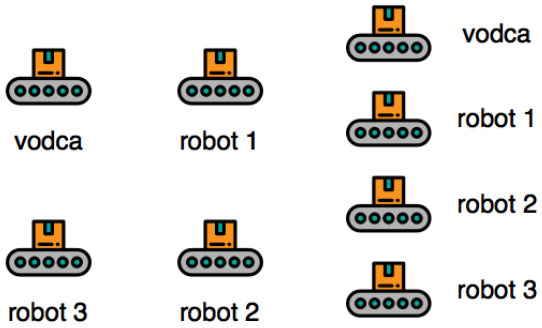
Obr. 6.2: Príklad mapy prostredia a popisného súboru k danej mape.

6.2 Program

Programom v tomto prípade rozumieme uzol, ktorý odoberá a publikuje dáta na konkrétne témy. Základom programu je odoberanie dát popisujúcich cestu skupiny robotov k zvolenému cielu. Tieto dáta sa spracujú, a pre každý robot sa z nich vypočíta pozícia kam sa má robot presunúť. Následne sa táto informácia publikuje na príslušnú tému, čo spôsobí, že robot sa presunie na danú pozíciu.

6.2.1 Algoritmus výpočtu pozície

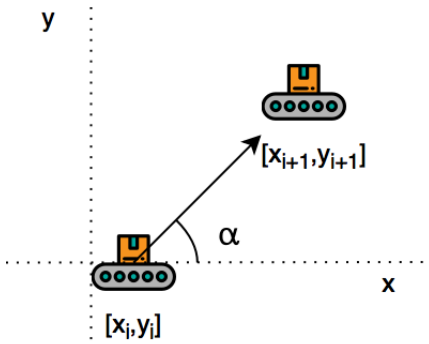
Základným tvarom formácie bude štvorec. Ak niektorý z robotov narazí na prekážku, štvorcová formácia sa zmení na formáciu tvaru rad. Na obrázku 6.3 je zobrazený základný štvorcový i radový tvar formácie s označením robotov, ktoré je použité vo zvyšku tejto práce.



Obr. 6.3: Základná štvorcová formácia a formácia tvaru rad.

Algoritmus výpočtu nových pozícií robotov počíta s tým, že dáta popisujúce trajektóriu skupiny robotov v skutočnosti predstavujú trajektóriu vodcu (leadera). Pozície následovníkov sa vypočítajú v niekoľkých krokoch. Najskôr sa vypočíta velkosť uhla α (obr. 6.4) medzi vodcovou súčasnou a jeho následujúcou pozíciou - tým zistíme, ktorým smerom je otočený. V následujúcich rovniciach značia x_i a y_i súradnice x a y súčasnej polohy vodcu, x_{i+1} a y_{i+1} značia súradnice x a y nasledujúcej polohy vodcu. $x1_i$ a $y1_i$ značia súradnice x a y súčasnej polohy robota 1, $x1_{i+1}$ a $y1_{i+1}$ značia súradnice x a y nasledujúcej polohy robota 1. Analogický zápis platí pre robotov 2 a 3. d značí vzdialenosť susedných robotov. Velkosť uhla α sa vypočíta rovnicou 6.1.

$$\alpha = \tan^{-1}\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right) \tag{6.1}$$



Obr. 6.4: Uhol α udáva uhol medzi súčasnou $[x_i, y_i]$ a následujúcou $[x_{i+1}, y_{i+1}]$ pozíciou vodcu.

Následne použitím tohto uhla a vodcovej následujúcej pozície vypočítame polohy pre následovníkov. Poloha robota 1, znázornená na obr. 6.5, sa vypočíta rovnicami 6.2.

$$\begin{aligned} d_{x1} &= \sin(\alpha) * d & x1_{i+1} &= x_{i+1} + d_{x1} \\ & & x1_{i+1} &= x_{i+1} + \sin(\alpha) * d \\ d_{y1} &= \cos(\alpha) * d & y1_{i+1} &= y_{i+1} - d_{y1} \\ & & y1_{i+1} &= y_{i+1} - \cos(\alpha) * d \end{aligned} \tag{6.2}$$

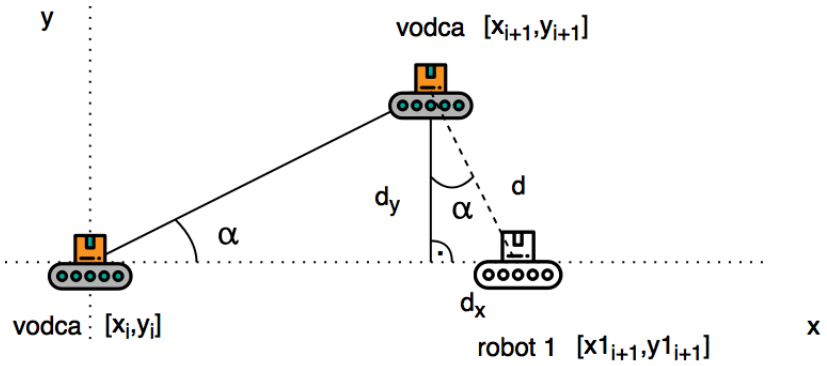
Pre výpočet polohy robota 2, obr. 6.6, je potrebné najskôr vypočítať polohu robota 1. Pre tento výpočet sú použité rovnice 6.3.

$$\begin{aligned} d_{x2} &= \cos(\alpha) * d & x2_{i+1} &= x1_{i+1} - d_{x2} \\ & & x2_{i+1} &= x1_{i+1} - \cos(\alpha) * d \\ d_{y2} &= \sin(\alpha) * d & y2_{i+1} &= y1_{i+1} - d_{y2} \\ & & y2_{i+1} &= y1_{i+1} - \sin(\alpha) * d \end{aligned} \tag{6.3}$$

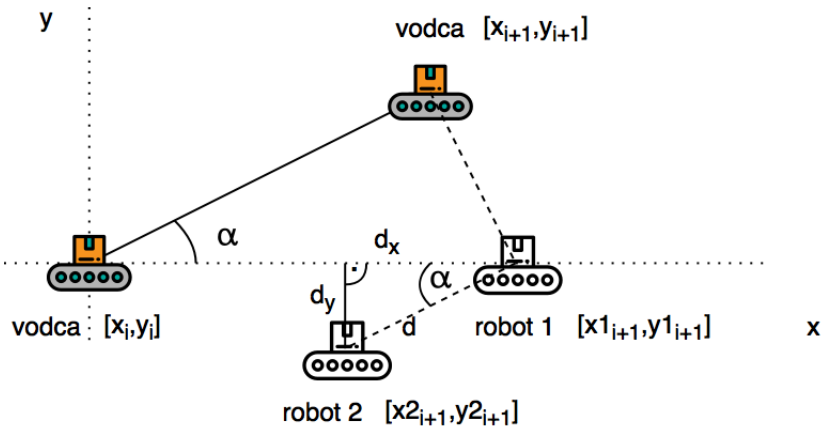
Dosadením rovnic 6.2 do 6.3 vzniknú rovnice 6.4.

$$\begin{aligned} x2_{i+1} &= x_{i+1} + \sin(\alpha) * d - \cos(\alpha) * d \\ y2_{i+1} &= y_{i+1} - \cos(\alpha) * d - \sin(\alpha) * d \end{aligned} \tag{6.4}$$

Na obr. 6.7 je zobrazená poloha robota 3. Rovnice pre jej výpočet sú 6.5.



Obr. 6.5: Určenie polohy robota 1 na základe následujúcej polohy vodcu.

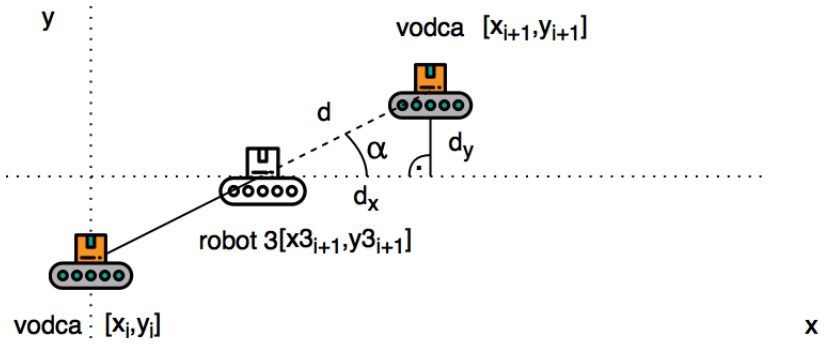


Obr. 6.6: Určenie polohy robota 2 na základe následujúcej polohy robota 1.

$$\begin{aligned} d_{x3} = \cos(\alpha) * d \qquad & x3_{i+1} = x_{i+1} - d_{x3} \\ & x3_{i+1} = x_{i+1} - \cos(\alpha) * d \\ d_{y3} = \sin(\alpha) * d \qquad & y3_{i+1} = y_{i+1} - d_{y3} \\ & y3_{i+1} = y_{i+1} - \sin(\alpha) * d \end{aligned} \tag{6.5}$$

Pri výpočte nasledujúcej pozície každého robota je potrebné zistiť, či sa na vypočítanej pozícií nenachádza prekážka. V prípade, že vypočítaná poloha je obsadená, dôjde k zmene formácie na tvar rad. Táto formácia je zobrazená na obrázku 6.8. Rovnice pre výpočet polohy jednotlivých robotov (6.6, 6.7, 6.8) sú obdobné rovniciam pre výpočet polohy robota 3 v štvorcovej formácii (6.5).

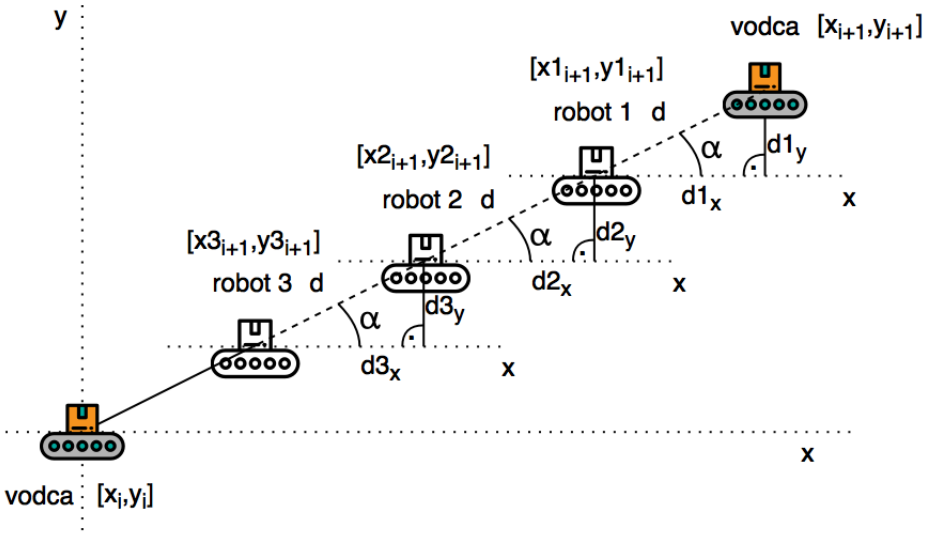
$$\begin{aligned} x1_{i+1} &= x_{i+1} - \cos(\alpha) * d \\ y1_{i+1} &= y_{i+1} - \sin(\alpha) * d \end{aligned} \tag{6.6}$$



Obr. 6.7: Určenie polohy robota 3 na základe následujúcej polohy vodcu.

$$\begin{aligned} x2_{i+1} &= x1_{i+1} - \cos(\alpha) * d \\ y2_{i+1} &= y1_{i+1} - \sin(\alpha) * d \\ x2_{i+1} &= x_{i+1} - 2 * (\cos(\alpha) * d) \\ y2_{i+1} &= y_{i+1} - 2 * (\sin(\alpha) * d) \end{aligned} \tag{6.7}$$

$$\begin{aligned} x3_{i+1} &= x2_{i+1} - \cos(\alpha) * d \\ y3_{i+1} &= y2_{i+1} - \sin(\alpha) * d \\ x3_{i+1} &= x_{i+1} - 3 * (\cos(\alpha) * d) \\ y3_{i+1} &= y_{i+1} - 3 * (\sin(\alpha) * d) \end{aligned} \tag{6.8}$$



Obr. 6.8: Určenie polohy robotov 1, 2 a 3 vo formácii tvaru rad.

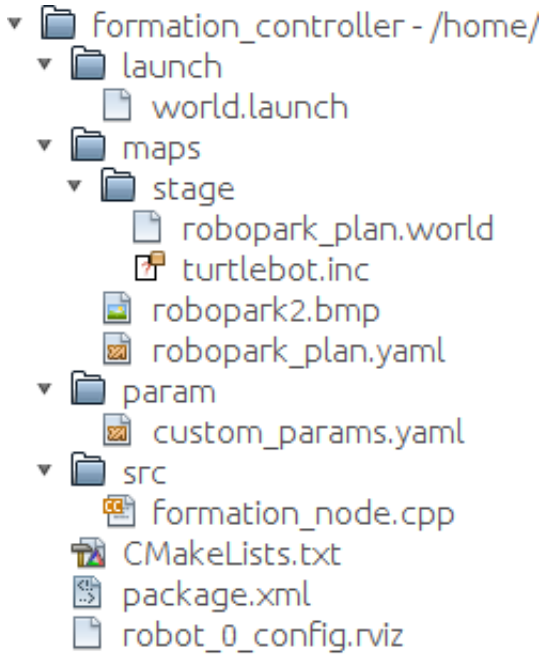
Kapitola 7

Implementované riešenie

V tejto kapitole bude podrobne popísaná implementácia programu a jednotlivých modulov potrebných na spustenie simulácie. Program je balíček ROS napísaný v jazyku C++. V súboroch potrebných pre spustenie simulácie sú použité jazyky XML a YAML.

7.1 Súborová štruktúra programu

Súborová štruktúra programu je zobrazená na obr. 7.1.



Obr. 7.1: Súborová štruktúra programu.

Súbor *CMakeLists.txt* je vstupom pre CMake buildovací systém pre vytvorenie softvérového balíčka. Súbor *package.xml* je manifest balíčka, definuje vlastnosti balíčka ako sú názov, číslo verzie, autor a závislosti na iných balíčkoch.

V zložke *maps* je bitmapový obrázok mapy (*robopark2.bmp*) a súbor *robopark_plan.yaml*, ktorý obsahuje metadáta o danej mape. Obsah tohto popisného súboru je zobrazený na obr. 7.2. Tento súbor obsahuje:

názov súboru s mapou i s cestou k nej,

rozlíšenie mapy - v metroch na pixel, teda rozlíšenie 0,02 znamená, že na jeden meter pripadá $\frac{1}{0,02} = 50$ pixelov,

súradnice počiatku mapy - 2D súradnice lavého dolného pixelu v mape, určené ako [x, y, uhol otočenia],

negatív - ak je mapa v negatívnej forme (hodnota 1), znamená to, že obsadené pixely sú zobrazené bielou farbou a voľné pixely čiernou,

prah obsadených pixelov - pri načítaní bitmapového obrázku sa určí pre každý pixel hodnota od 0 do 1 na základe jeho farby. Prah obsadených pixelov je hodnota v rozmedzí 0 až 1 určujúca, že pixely s hodnotou rovnou alebo vyššiou ako je táto hodnota sú považované za obsadené,

prah voľných pixelov - hodnota 0 až hodnota prahu obsadených pixelov určujúca, že pixely s hodnotou nižšou ako je táto hodnota sú považované za neobsadené.

```
image: robopark2.bmp
resolution: 0.02
origin: [0, 0, 0.0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

Obr. 7.2: Obsah súboru *robopark_plan.yaml* obsahujúci metadáta k danej mape.

Súbor *robopark_plan.world* definuje opäť mapu. Túto mapu využíva na simuláciu balíček *stage_ros*. V tomto súbore sú tiež prepísané niektoré vlastnosti každého robota ako napr. počiatočná poloha, meno a farba (obr. 7.3).

```
include "turtlebot.inc"

# throw in a robot
turtlebot
(
  pose [ 3.0 6.0 0.0 0.0 ]
  name "leader"
  color "black"
)
```

Obr. 7.3: Definícia niektorých vlastností robota v súbore *robopark_plan.world*.

Všetky vlastnosti robota sú definované v súbore *turtlebot.inc* na obr. 7.4. Medzi tieto vlastnosti patrí napr.:

senzor - definícia vlastností použitého senzoru, napr. maximálny dosah, vzorkovanie alebo jeho veľkosť,

počiatočná pozícia robota , ktorá je ale prepísaná v súbore *robopark_plan.world*,

rozmery robota v tvare 3D súradníc (x, y, z),

umiestnenie senzoru - súradnice v tvare (x, y, z, uhol otočenia)

```
define kinect ranger
(
  sensor
  (
  range_max 6.5
    fov 58.0
    samples 640
  )
  # generic model properties
  color "black"
  size [ 0.06 0.15 0.03 ]
)

define turtlebot position
(
  pose [ 0.0 0.0 0.0 0.0 ]
  odom_error [0 0 0 0 0 0]
  size [ 0.2552 0.2552 0.40 ]
  origin [ 0.0 0.0 0.0 0.0 ]
  gui_nose 1
  drive "diff"
  color "gray"
  kinect(pose [ -0.1 0.0 -0.11 0.0 ])
)
```

Obr. 7.4: Definícia základných vlastností robota v súbore *turtlebot.inc*.

V súbore *custom-params.yaml* (na obr. 7.5) sú definované parametre uzlov z *world.launch* súboru, ktoré potrebujeme prepísať. Ide o maximálnu rýchlosť vodcu a jeho zrýchlenie. Keďže vodca nemá spätnú väzbu od následovníkov, znížili sme jeho rýchlosť a zrýchlenie tak, aby mal v prípade chyby každý následovník možnosť navrátiť sa do formácie.

```
DWAPlannerROS:
  max_vel_x: 0.1
  acc_lim_x: 0.5
```

Obr. 7.5: Obsah súboru *custom-params.yaml*, v ktorom sú definované tie parametre uzlov z *world.launch* súboru, ktoré potrebujeme prepísať.

V súbore *robot_0_config.rviz* sú uložené dáta pre balíček *rviz* určujúce témy, ktoré majú byť zobrazované počas behu simulácie. Zobrazované témy sú mapa, poloha a trajektória jednotlivých robotov.

Súbor *world.launch* je spúšťacím súborom celého programu. Je to XML súbor obsahujúci informácie o všetkých uzloch, ktoré sa majú spustiť, aby simulácia fungovala. Tieto uzly sú balíčky pre vizualizáciu simulácie *stage_ros* a *rviz* (obr. 7.6), nami vytvorený balíček *formation_controller* a pre každý robot zvlášť:

```
<node name="rviz" pkg="rviz" type="rviz" args="-d $(find formation_controller)/robot_0_config.rviz">
  <remap from="/move_base_simple/goal" to="move_base_simple/goal" />
</node>
```

Obr. 7.6: Definícia v súbore *world.launch* pre spustenie balíčka *rviz* s požadovanou konfiguráciou.

model robota - na získanie modelu robota sme použili balíček *turtlebot_bringup*. Parametrami tohto balíčka sme definovali vybraného robota *kobuki* s hexagonálnou podstavou a senzorom kinect (obr. 7.7),

```
<arg name="base"       default="$(optenv TURTLEBOT_BASE kobuki)"/>  <!-- create, rhoomba -->
<arg name="stacks"     default="$(optenv TURTLEBOT_STACKS hexagons)"/>  <!-- circles, hexagons -->
<arg name="3d_sensor"  default="$(optenv TURTLEBOT_3D_SENSOR kinect)"/>  <!-- kinect, asus_xtion_pro -->
 <include file="$(find turtlebot_bringup)/launch/includes/description.launch.xml">
   <arg name="base" value="$(arg base)" />
   <arg name="stacks" value="$(arg stacks)" />
   <arg name="3d_sensor" value="$(arg 3d_sensor)" />
 </include>
```

Obr. 7.7: Definícia v súbore *world.launch* pre spustenie balíčka *turtlebot_bringup* s požadovanou konfiguráciou.

informácie o stave robota - balíček *robot_state_publisher*,

diagnostika robota - balíček *diagnostic_aggregator*,

navigácia - balíček *turtlebot_navigation* na obrázku 7.8,

```
<!--  ************** Navigation  ***************  -->
<include file="$(find turtlebot_navigation)/launch/includes/move_base.launch.xml">
  <arg name="global_frame_id" value="/map"/>
  <arg name="odom_frame_id" value="robot_0/odom"/>
  <arg name="base_frame_id" value="robot_0/base_footprint"/>
  <arg name="odom_topic" value="robot_0/odom"/>
  <arg name="laser_topic" value="robot_0/scan"/>
  <arg name="custom_param_file" value="$(find formation_controller)/param/custom_params.yaml"/>
</include>
```

Obr. 7.8: Definícia v súbore *world.launch* pre spustenie balíčka *turtlebot_navigation* s požadovanou konfiguráciou.

mapa - balíček *map_server*, ktorému sa predáva odkaz na *robopark_plan.yaml* ako parameter (obr. 7.9),

transformácie - balíček *tf*, ktorý slúži na publikovanie transformácií z lokálneho súradnicového systému do globálneho.

```
<arg name="map_file"       default=" $(find formation_controller)/maps/robopark_plan.yaml"/>
 <!--  ****** Maps *****  -->
 <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)">
   <param name="frame_id" value="/map"/>
 </node>
```

Obr. 7.9: Definícia v súbore *world.launch* pre spustenie balíčka *turtlebot_bringup* s požadovanou konfiguráciou.

Súbor *formation_node.cpp* obsahuje program s logikou výpočtu pozícií následovníkov. Vstupným bodom programu, ktorý je napísaný v jazyku C++, je funkcia main. Táto funkcia obsahuje okrem štandardných inicializačných príkazov frameworku ROS definície publisherov a subscriberov - zobrazené na obr. 7.10.

```
// publishers used to set goals for following robots
follower_1_Publisher = nh.advertise<geometry_msgs::PoseStamped>("/robot_1/move_base_simple/goal", 10);
follower_2_Publisher = nh.advertise<geometry_msgs::PoseStamped>("/robot_2/move_base_simple/goal", 10);
follower_3_Publisher = nh.advertise<geometry_msgs::PoseStamped>("/robot_3/move_base_simple/goal", 10);

// subscriber to leaders local plan
leaderGoalSubscriber = nh.subscribe<nav_msgs::Path>("/robot_0/move_base/DWAPlannerROS/local_plan", 10,
            calculateNewFollowerCoordinates);
// subscriber to global costmap
mapSubscriber = nh.subscribe<nav_msgs::OccupancyGrid>("/robot_0/move_base/global_costmap/costmap", 10,
            saveMap);
// subscriber to transformation messages - used to gain leader's initial position
initialPositionOfLeaderSubscriber = nh.subscribe<tf2_msgs::TFMessage>("/tf", 10, initialPose);
```

Obr. 7.10: Publishery a subsribery použití vo *formation_node.cpp*.

Publishery slúžia na publikovanie vypočítanej pozície daného robota na tému */robot_* + číslo robota + */move_base_simple/goal*, čo spôsobí, že robot s daným číslom sa posunie na túto pozíciu.

Subscriber *mapSubscriber* odoberá správy z témy */robot_0/move_base/global_costmap/costmap*, na ktorú sú neustále publikované správy o globálnej costmape. Sú to informácie o pravdepodobnosti obsadenosti jednotlivých pixelov celej mapy. Keďže algoritmus pracuje v statickom prostredí, teda okrem algoritmom ovládaných robotov sa v prostredí nič iné nepohybuje, stačí túto mapu načítať iba raz (obr. 7.11).

```
void saveMap(nav_msgs::OccupancyGrid newMap) {
    if (!isMapRead) {
        currentMap = newMap;

        isMapRead = true;
    }
}
```

Obr. 7.11: Funkcia *saveMap* zo súboru *formation_node.cpp*.

Subscriber *initialPositionOfLeaderSubscriber* odoberá správy z témy */tf*, na ktorú sú publikované správy s transformáciami z lokálneho súradnicového systému jednotlivých uzlov do globálneho systému. Po spustení aplikácie patrí počiatočná poloha jednotlivých robotov medzi prvé správy, ktoré sa na tejto téme objavia. Algoritmus zaujíma iba počiatočná poloha vodcu skupiny, ktorú si uloží a bude ju používať v ďalších výpočtoch (obr. 7.12).

Subscriber *leaderGoalSubscriber* odoberá správy obsahujúce vodcovu naplánovanú cestu k cieľu. Tieto správy odoberá z témy */robot_0/move_base/DWAPlannerROS/local_plan* a sú spracovávané vo funkcii *calculateNewFollowerCoordinates*. V tejto funkcii sa najskôr

```
void initialPose(tf2_msgs::TFMessage msg) {
    if (!initialPositionRead) {
        if (msg.transforms[0].child_frame_id == "robot_0/odom") {
            if (msg.transforms[0].header.frame_id == "/map") {

                initialLeaderPose.position.x = msg.transforms[0].transform.translation.x;
                initialLeaderPose.position.y = msg.transforms[0].transform.translation.y;

                initialPositionRead = true;
            }
        }
    }
}
```

Obr. 7.12: Funkcia *initialPose* zo súboru *formation_node.cpp*.

podla rovnice 6.1 vypočíta uhol α (obr. 7.13) a následne sa počítajú koordináty pre jednotlivé roboty. *theta* je uhol natočenia vodcu voči osi x. V algoritme je tento uhol využitý na určenie, do ktorého kvadrantu smeruje vodca. V prípade, že vodca smeruje do 2. alebo 3. kvadrantu ($theta > \frac{\pi}{2}$), je potrebné invertovať znamienko sin a cos v rovniciach 6.2 až 6.8.

```
double tangent = (pathToGoal->poses[i + 1].pose.position.y - pathToGoal->poses[i].pose.position.y)
        / (pathToGoal->poses[i + 1].pose.position.x - pathToGoal->poses[i].pose.position.x);
double angle = atan(tangent);
double sinus = sin(angle);
double cosinus = cos(angle);
double theta = 2 * tf2::angle(tf2::Quaternion(pathToGoal->poses[i].pose.orientation.x,
        pathToGoal->poses[i].pose.orientation.y, pathToGoal->poses[i].pose.orientation.z,
        pathToGoal->poses[i].pose.orientation.w), unityQuat);
```

Obr. 7.13: Výpočet uhla α z rovnice 6.1 v súbore *formation_node.cpp*.

Na obrázku 7.14 je znázornený výpočet polohy robota 1 podla rovnice 6.2 s prihliadnutím na to, do ktorého kvadrantu vodca smeruje. Vzdialenosť medzi robotmi d je jeden meter. Po vypočítaní koordinátov pozície následovníka sa volaním funkcie *isThereObstacle* testuje, či nie je daná pozícia obsadená.

```
if (theta > (PI / 2)) {
    x_follower = x_leaderT - sinus;
    y_follower = y_leaderT + cosinus;
} else {
    x_follower = x_leaderT + sinus;
    y_follower = y_leaderT - cosinus;
}

if (isThereObstacle(x_follower, y_follower)) {
    obstacle = true;
} else {
    follower_1_Goal.pose.position.x = x_follower;
    follower_1_Goal.pose.position.y = y_follower;
}
```

Obr. 7.14: Výpočet polohy robota 1 podla rovnice 6.2 v súbore *formation_node.cpp*.

Vstupom funkcie *isThereObstacle* sú x a y koordináty pozície, ktorú treba otestovať na prítomnosť prekážky. Tieto koordináty sú v lokálnom súradnicovom systéme vodcu, preto je nutné ich pri výpočte zosúladiť s počiatkom costmapy. Výsledné koordináty sa delia rozlíšením mapy, aby sa získal index konkrétneho pixelu, ktorý zodpovedá zadaným koordinátom. Výsledkom je ale číslo typu *double* a tak je potrebné pretypovať toto číslo

na *integer*. V C++ pri pretypovaní čísla z double na integer dôjde iba k odseknutiu desatinnej časti čísla, preto je pre väčšiu presnosť pripočítaná hodnota 0,5 pred pretypovaním. Na obr. 7.15 je zobrazený tento výpočet.

```
// x_position and y_position are relative to leader's initial position
x_position += initialLeaderPose.position.x;
y_position += initialLeaderPose.position.y;

// transform given x and y position to global costmap origin (costmap's origin does not need to be 0,0)
// then divide it by map resolution to get specific grid cell in costmap
// double is truncated to integer - add 0.5 to get more precise result
int x_pos = (unsigned int) ((x_position - currentMap.info.origin.position.x)
        / currentMap.info.resolution + 0.5);
int y_pos = (unsigned int) ((y_position - currentMap.info.origin.position.y)
        / currentMap.info.resolution + 0.5);
```

Obr. 7.15: Výpočet 2D indexov pixelu v mape zodpovedajúceho vstupným x a y koordinátom.

Costmapa je uložená v premennej typu jednorozmerné pole, preto je potrebné z 2D indexov pixelu, vypočítaných v predchádzajúcom kroku, vypočítať index. Postup je na obr. 7.16.

```
// The map data are stored in 1D array in row-major order
int index = y_pos * width + x_pos;
```

Obr. 7.16: Výpočet indexu pixelu v mape z 2D indexov toho istého pixelu.

Po vypočítaní indexu algoritmus zisťuje, či hodnota pixelu s daným indexom je menšia ako hodnota prahu voľných pixelov (obr. 7.2). Ak je menšia, tento pixel je voľný.

Ak aspoň jedna vypočítaná pozícia následovníkov je obsadená, vypočítajú sa nové súradnice pre pozície každého robota tak, aby výsledný tvar formácie bol rad. Výpočet pre robota 1 je zobrazený na obr. 7.17. Pozícia každého ďalšieho robota sa počíta rovnakým spôsobom, pri čom sa vychádza z pozície predchádzajúceho robota.

```
if (theta > (PI / 2)) {
    x_follower = x_leaderT + cosinus * spacing;
    y_follower = y_leaderT + sinus * spacing;
} else {
    x_follower = x_leaderT - cosinus * spacing;
    y_follower = y_leaderT - sinus * spacing;
}

follower_1_Goal.pose.position.x = x_follower;
follower_1_Goal.pose.position.y = y_follower;
```

Obr. 7.17: Výpočet súradníc pozície robota 1 vo formácii typu rad.

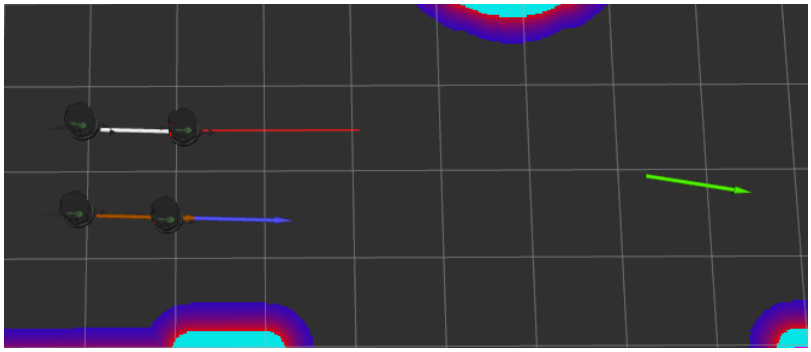
Vypočítané pozície pre každý robot sa v poslednom kroku algoritmu publikujú na príslušné témy (obr. 7.18). Frekvencia publikácie je 1 Hz. Túto hodnotu sme získali experimentovaním. Vyššie hodnoty frekvencie publikovania spôsobovali to, že roboty dostávali správy rýchlejšie ako na ne dokázali reagovať, čo spôsobovalo, že robot ostal stáť na mieste. Nižšie frekvencie spôsobovali to, že časový interval medzi zaslaním dvoch správ bol výrazne väčší ako čas potrebný pre vykonanie reakcie na danú správu, čo spôsobovalo, že sa roboty zastavovali a tým sa rozpadol konzistentný tvar formácie.

```
// loop for publishing goals for followers
while (ros::ok()) {

    if (newGoal) {
        follower_1_Publisher.publish(follower_1_Goal);
        follower_2_Publisher.publish(follower_2_Goal);
        follower_3_Publisher.publish(follower_3_Goal);
        newGoal = false;
        loop_rate.sleep();
    }

    ros::spinOnce();

}
```

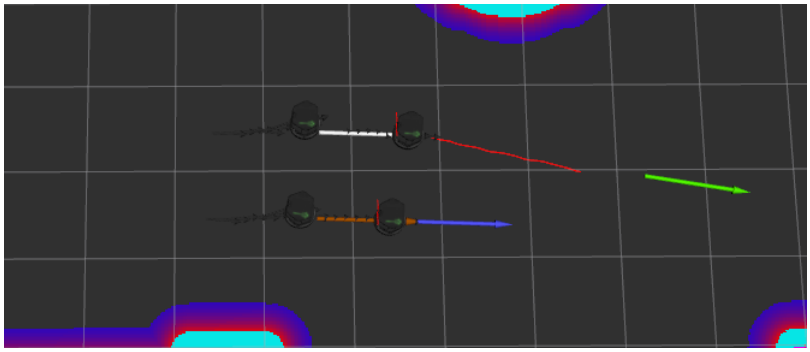
Obr. 7.18: Publikovanie súradníc pozícií každého následovníka.

7.2 Ukážka simulácie

Simulácia č. 1 (obr. 7.19, 7.20 a 7.21) ukazuje pohyb formácie po ceste bez prekážky. Na obr. 7.20 je vidno, že roboty udržujú štvorcový tvar formácie i počas pohybu.

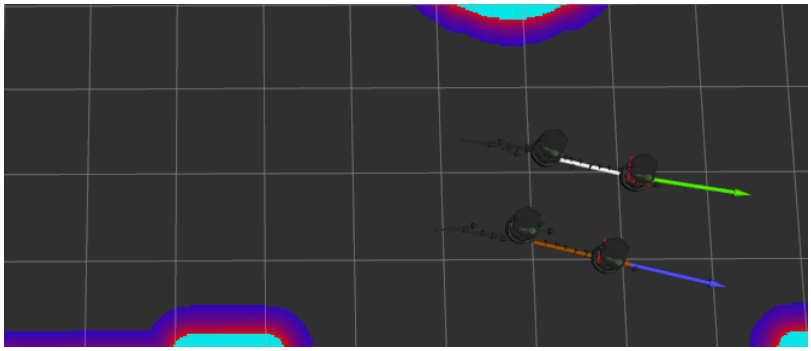


Obr. 7.19: Simulácia č. 1: pohyb formácie po ceste bez prekážky. Počiatočná pozícia.



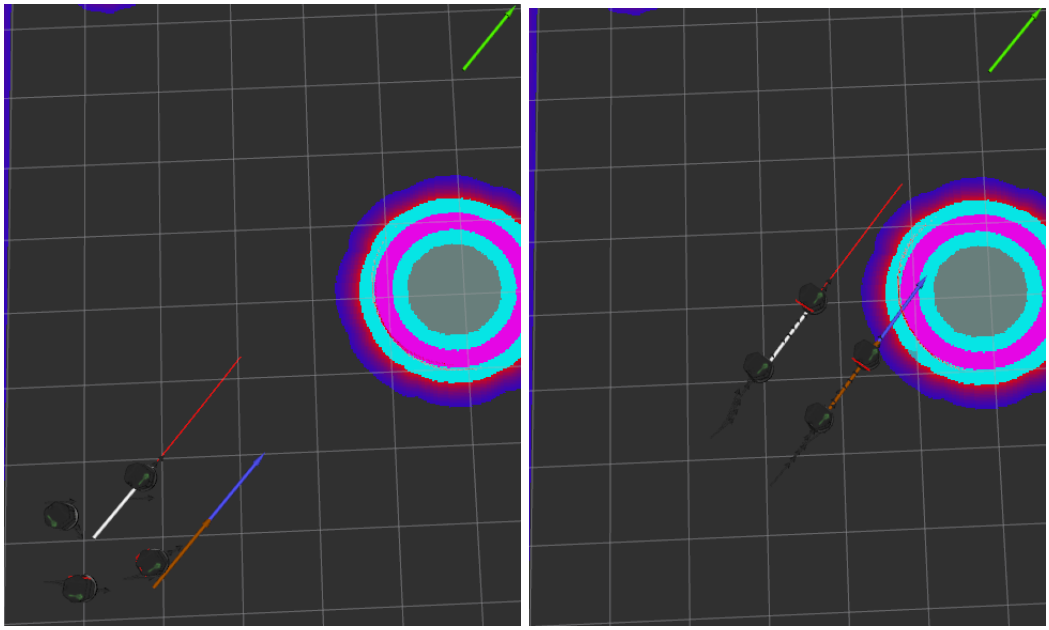
Obr. 7.20: Simulácia č. 1: pohyb formácie po ceste bez prekážky. Formácia v pohybe.

Simulácia č. 2 (obr. 7.22, 7.23 a 7.24) zobrazuje pohyb formácie po ceste s prekážkou. Na obr. 7.23 je vidieť, že algoritmus zistil, že v prípade štvorcovej formácie pozícia robota 1 koliduje s prekážkou, a preto došlo k zmene formácie na tvar rad. Na obr. 7.24 je vidno,

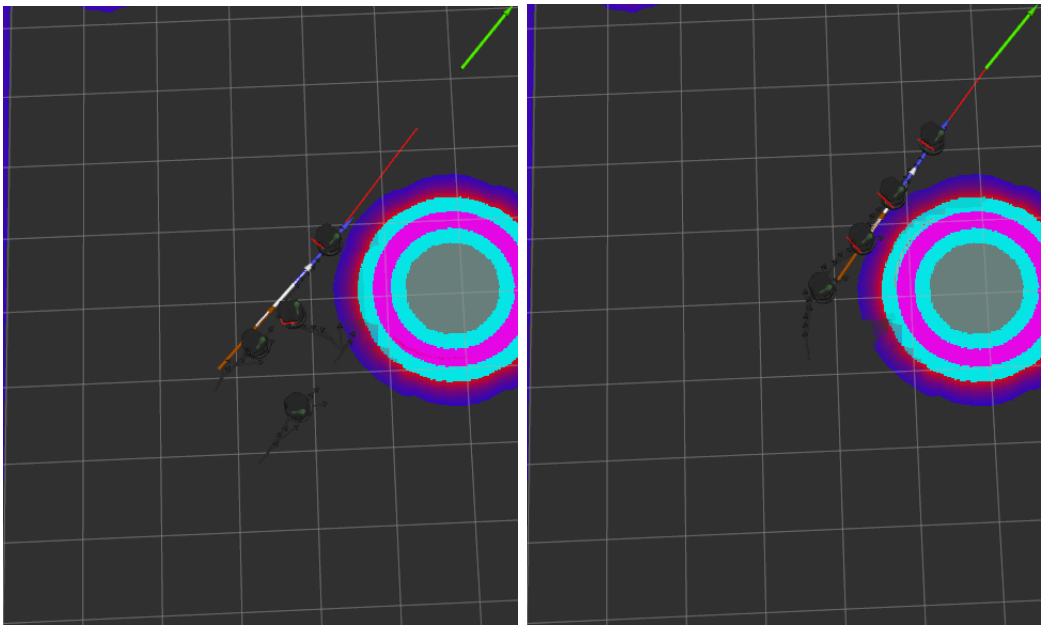


Obr. 7.21: Simulácia č. 1: pohyb formácie po ceste bez prekážky. Formácia dorazila do ciela.

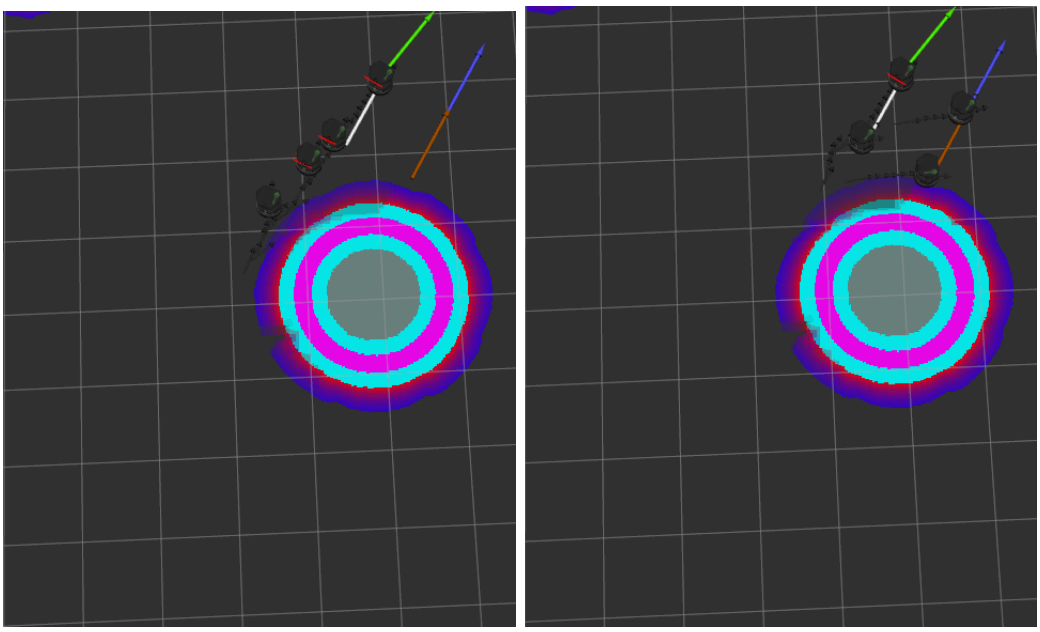
že formácia zmenila tvar na štvorec akonáhle to bolo opäť možné - teda pozícia robota 2 vo štvorcovej formácii bola volná.



Obr. 7.22: Simulácia č. 2: pohyb formácie po ceste s prekážkou. Formácia musí zmeniť svoj tvar počas cesty ku cielu (1. časť).



Obr. 7.23: Simulácia č. 2: pohyb formácie po ceste s prekážkou. Formácia musí zmeniť svoj tvar počas cesty ku cielu (2. časť).



Obr. 7.24: Simulácia č. 2: pohyb formácie po ceste s prekážkou. Formácia musí zmeniť svoj tvar počas cesty ku cielu (3. časť).

Kapitola 8

Záver

V tejto práci sú predstavené niektoré existujúce algoritmy pre plánovanie cesty pre formáciu robotov so zameraním sa na riešenie problematiky udržania formácie počas celého presunu formácie k cielu. Ďalej je v práci predstavený vlastný algoritmus pre plánovanie cesty pre formáciu robotov založený práve na existujúcich algoritmoch predstavených v prvej časti práce.

Pre plánovanie cesty jedného robota existuje množstvo rôznych algoritmov, no pre plánovanie cesty pre formáciu robotov tomu tak nie je. Preto sa táto práca zaoberá práve touto problematikou.

Nami navrhnutý algoritmus využíva práve toho, že existuje množstvo plánovacích algoritmov pre jedného robota. Cieľom tejto práce nie je znovu vymyslieť koleso, preto sme si ako základ nášho algoritmu vzali plánovací algoritmus, ktorý je vo frameworku ROS už implementovaný. Globálny plánovač tohto frameworku v základnom nastavení používa Dijkstrov algoritmus (v zdrojovom kóde globálneho plánovača sa nachádza i implementácia optimalizovaného A* algoritmu, ale je v ňom chyba a ešte ju nestihli opraviť). Cesta vypočítaná týmto plánovačom je spracovaná navrhnutým algoritmom a výstupom je taká cesta pre jednotlivé roboty skupiny, ktorá zaisťuje udržanie tvaru formácie.

V nadväzujúcej práci by mala byť vyriešená slabina navrhnutého algoritmu, ktorou je jeho funkčnosť iba v statickom prostredí. Roboty by mali byť vybavené senzormi na sledovanie svojho okolia a algoritmus by mal byť upravený tak, aby reagoval v reálnom čase na dáta prichádzajúce z týchto senzorov. Ďalším smerom vývoja by mohlo byť upravenie algoritmu tak, aby hladal prioritne takú cestu k cielu, ktorou by prešla formácia robotov bez potreby zmeny svojho tvaru. Až v prípade, že by takúto cestu nenašiel, vyhladal by najkratšiu cestu k cielu. Taktiež optimalizácia procesu zmeny formácie je jednou z oblastí, na ktorú by sa nadväzujúca práca mala zamerať.

Literatúra

[1] *Coppelia Robotics V-REP: Create. Compose. Simulate. Any Robot.* [Online; navštíveno 23.01.2017].
URL http://www.coppeliarobotics.com/index.html

[2] *Documentation - ROS Wiki. ROS.org.* [Online; navštíveno 04.12.2016].
URL http://wiki.ros.org//

[3] *Gazebo: Tutorial: Beginner: Overview.* [Online; navštíveno 23.01.2017].
URL http://gazebosim.org/tutorials?cat=guided_b&tut=guided_b1

[4] *ROS Carrot Planner.* [Online; navštíveno 14.05.2017].
URL http://wiki.ros.org/carrot_planner

[5] *ROS Dynamic Window Approach (DWA) Planner.* [Online; navštíveno 14.05.2017].
URL http://wiki.ros.org/dwa_local_planner

[6] Abichandani, P.; Benson, H. Y.; Kam, M.: Multi-vehicle path coordination in support of communication. In *2009 IEEE International Conference on Robotics and Automation, ICRA 2009, Kobe, Japan, May 12-17, 2009*, IEEE, 2009, s. 3237–3244.
URL http://dx.doi.org/10.1109/ROBOT.2009.5152787

[7] Badgerow, J. P.: An analysis of function in the formation flight of canadian geese. *The Auk*, ročník 105, 1988: s. 749 – 755.

[8] Choset, H. M.; Lynch, K. M.; Hutchinson, S.; aj.: *Principles of robot motion: theory, algorithms, and implementation.* Intelligent robotics and autonomous agents, Cambridge (Mass.), London: MIT Press, ISBN 0-262-03327-5, 40 s.

[9] Choset, H. M.; Lynch, K. M.; Hutchinson, S.; aj.: *Principles of robot motion: theory, algorithms, and implementation.* Intelligent robotics and autonomous agents, Cambridge (Mass.), London: MIT Press, ISBN 0-262-03327-5, 254 - 256 s.

[10] Consolini, L.; Morbidi, F.; Prattichizzo, D.; aj.: Leader–follower formation control of nonholonomic mobile robots with input constraints. *Automatica*, ročník 44, č. 5, 2008: s. 1343 – 1349, ISSN 0005-1098,
doi:http://dx.doi.org/10.1016/j.automatica.2007.09.019.
URL http://www.sciencedirect.com/science/article/pii/S0005109807004578

[11] Defoort, M.; Floquet, T.; Kokosy, A.; aj.: Sliding-Mode Formation Control for Cooperative Autonomous Mobile Robots. *IEEE Transactions on Industrial Electronics*, ročník 55, č. 11, Nov 2008: s. 3944 – 3953, ISSN 0278-0046,
doi:10.1109/TIE.2008.2002717.
URL http://ieeexplore.ieee.org.ezproxy.lib.vutbr.cz/document/4601469/

[12] Dong, W.; Farrell, J. A.: Cooperative Control of Multiple Nonholonomic Mobile Agents. *IEEE Transactions on Automatic Control*, ročník 53, č. 6, Jul 2008: s. 1434 – 1448, ISSN 0018-9286, doi:10.1109/TAC.2008.925852.
URL http://ieeexplore.ieee.org.ezproxy.lib.vutbr.cz/document/4610021/

[13] Fox, D.; Burgard, W.; Kruppa, H.; aj.: *A probabilistic approach to collaborative multi-robot localization. Auton. Robots*, ročník 8, č. 3, 2000: s. 325–344.

[14] Franchi, A.; Freda, L.; Oriolo, G.; aj.: *The sensor-based random graph method for cooperative robot exploration. IEEE/ASME Trans. Mechatronics*, ročník 14, č. 2, 2009: s. 163–175.

[15] Krcek, P.: *Plánování cesty autonomního lokomočního robotu na základě strojového učení.* Dizertační práce, Fakulta strojního inženýrství ústav automatizace a informatiky, Vysoké učení technické, 2010, s. 13.

[16] LaValle, S. M.; Kuffner, J. J.; Jr.: Rapidly-Exploring Random Trees: Progress and Prospects. 2000.

[17] Liu, S.; Sun, D.; Zhu, C.: Coordinated Motion Planning for Multiple Mobile Robots Along Designed Paths With Formation Requirement. *IEEE/ASME Transactions on Mechatronics*, ročník 16, č. 6, 2011: s. 1021 – 1031.

[18] Mastellone, S.; Stipanović, D. M.; Graunke, C. R.; aj.: Formation Control and Collision Avoidance for Multi-agent Non-holonomic Systems: Theory and Experiments. *The International Journal of Robotics Research*, ročník 27, č. 1, 2008: s. 107–126, doi:10.1177/0278364907084441.
URL http://dx.doi.org/10.1177/0278364907084441

[19] McFarland, D.: *Animal Behavior*. Benjamin Cummings, 1985.

[20] Naffin, D. J.: *Multi-Robot Formations: Rule-Based Synthesis And Stability Analysis.* Dizertační práce, Faculty Of The Graduate School University Of Southern California, 2006, s. 5.

[21] Naffin, D. J.: *Multi-Robot Formations: Rule-Based Synthesis And Stability Analysis.* Dizertační práce, Faculty Of The Graduate School University Of Southern California, 2006, s. 10.

[22] Quigley, M.; Conley, K.; Gerkey, B. P.; aj.: ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, 2009.

[23] Ren, W.: Consensus Tracking Under Directed Interaction Topologies: Algorithms and Experiments. *IEEE Transactions on Control Systems Technology*, ročník 18, č. 1, Jan 2010: s. 230 – 237, ISSN 1063-6536, doi:10.1109/TCST.2009.2015285.
URL http://ieeexplore.ieee.org.ezproxy.lib.vutbr.cz/document/5229136/

[24] Ren, W.; Sorensen, N.: Distributed coordination architecture for multi-robot formation control. *Robotics and autonomous systems*, ročník 56, č. 4, 2008: s. 324–333, ISSN 09218890, doi:10.1016/j.robot.2007.08.005.
URL http://www.sciencedirect.com.ezproxy.lib.vutbr.cz/science/article/pii/S0921889007001108

[25] Reynolds, C.: *Flocks, herds and schools: A distributive behavioral model. Computer Graphics*, ročník 21, č. 4, 1987: s. 25 – 34.

[26] Rezaee, H.; Abdollahi, F.: A Decentralized Cooperative Control Scheme With Obstacle Avoidance for a Team of Mobile Robots. *IEEE Transactions on Industrial Electronics*, ročník 61, č. 1, Jan 2014: s. 347–354, ISSN 0278-0046, doi:10.1109/TIE.2013.2245612.

[27] Shao, J.; Xie, G.; Wang, L.: Leader-following formation control of multiple mobile vehicles. *IET Control Theory & Applications*, ročník 1, č. 2, Mar 2007: s. 545 – 552, ISSN 1751-8644, doi:10.1049/iet-cta:20050371.
URL http://ieeexplore.ieee.org.ezproxy.lib.vutbr.cz/document/4123995/

[28] Tang, Z.; Ozguner, U.: *Motion planning for multi-target surveillance with mobile sensor agents. IEEE Trans. Robot.*, 2005: s. 898–908.

[29] Voyles, R.; Larson, A.: *TerminatorBot: A novel robot with dual-use mechanism for locomotion and manipulation. IEEE/ASME Trans. Mechatronics*, ročník 10, č. 1, 2005: s. 17–25.

[30] Wang, Y.; Silva, C.: *Sequential Q-learning with Kalman filtering for multirobot cooperative transportation. IEEE/ASME Trans. Mechatronics*, ročník 15, č. 2, 2010: s. 261–268.

[31] Whitehead, H.: *Encyclopedia of Marine Mammals*. Academinc Press, 2002.

[32] Xiao, F.; Wang, L.; Chen, J.; aj.: Finite-time formation control for multi-agent systems. *Automatica*, ročník 45, č. 11, 2009: s. 2605 – 2611, ISSN 0005-1098, doi:http://dx.doi.org/10.1016/j.automatica.2009.07.012.
URL //www.sciencedirect.com/science/article/pii/S0005109809003586

[33] Yang, P.; Freeman, R. A.; Lynch, K. M.: Multi-Agent Coordination by Decentralized Estimation and Control. *IEEE Transactions on Automatic Control*, ročník 53, č. 11, Dec 2008: s. 2480–2496, ISSN 0018-9286, doi:10.1109/TAC.2008.2006925.
URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4700861