VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

NÁVRH A REALIZACE STRATEGIE PLÁNOVANÍ POHYBU MOBILNÍHO ROBOTU

AUTONOMOUS MOBILE ROBOT MOTION PLANNING STRATEGY DESIGN

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. Miroslav Doseděl

VEDOUCÍ PRÁCE

SUPERVISOR

doc. Ing. Stanislav Věchet, Ph.D.

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Bc. Miroslav Doseděl**
Studijní program:	Mechatronika
Studijní obor:	bez specializace
Vedoucí práce:	**doc. Ing. Stanislav Věchet, Ph.D.**
Akademický rok:	2021/22

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

Návrh a realizace strategie plánovaní pohybu mobilního robotu

Stručná charakteristika problematiky úkolu:

Překládaná práce je věnována návrhu a realizaci lokálního plánovacího algoritmu pro mobilního robota určeného pro provoz ve vnitřním prostředí budov s cílem zalévání pokojových rostlin. Cílem práce je návrh algoritmu pro optimalizaci pohybu v blízkosti rostlin a nalezení vhodné trajektorie z hlediska dávkování přísunu vody. Součástí práce je také příprava algoritmu pro nájezd do optimální pozice vzhledem k rostlině, či dokovací stanici tak, aby bylo možné v budoucnu automaticky doplňovat vodu či nábijet akumulátory.

Cíle diplomové práce:

1) Navrhněte architekturu plánování pohybu robota během zálévání pokojových rostlin.
2) Navrhněte metodu globálního plánování pohybu robota v prostoru.
3) Navrhněte metodu lokálního plánování pohybu v oblasti zalévání.
4) Navrhněte metodu tzv. mikroplánování kolem vybrané rostliny pro navigaci do požadované zalévaci pozice.
5) Navržené algoritmy integrujte do řidicího systému robota.
6) Ověřte a zhodnoťte funcionalitu vybraného řešení.

Seznam doporučené literatury:

LIGHT, R. A.: Mosquitto: server and client implementation of the MQTT protocol, The Journal of Open Source Software, vol. 2, no. 13, May 2017, DOI: 10.21105/joss.00265.

BALÁTĚ, J.: Technické prostředky automatického řízení. Praha, SNTL 1986.

ROS.org. ROS.org | Powering the world's robots. [online]. 2.11.2016 [cit. 2016-11-02]. Dostupné z: http://www.ros.org/.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2021/22

V Brně, dne

L. S.

prof. Ing. Jindřich Petruška, CSc. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Tato práce se zabývá plánováním pohybu robotu v prostoru a v blízkosti rostlin. Hlavním cílem je realizovat pohyb robota určeného pro provoz ve vnitřních prostorách budov pro zalévání pokojových rostlin. V první části práce jsou rozebrány existující algoritmy pro plánování pohybu a vyhýbání se překážkám. V další části je napsána rešerše zabývající se Frameworkem ROS. Následně jsou popsány jednotlivé pohyby, které robot vykonává a na závěr je provedeno otestování na reálném robotu.

ABSTRACT

This work is concerned with planning the movement of the robot in space and near plants. The main goal is to realize the movement of a robot designed for operation in indoor areas of buildings for watering indoor plants. In the first part of the thesis, existing algorithms for motion planning and obstacle avoidance are presented. In the next part, a search dealing with Framework ROS is written. The individual movements performed by the robot are then described and testing on the real robot is carried out at the end.

KLÍČOVÁ SLOVA

Mobilní robot, plánování pohybu, globální plánovač, lokální plánovač, vyhýbání se překážkám, Framework ROS, Stage, Gazebo

KEYWORDS

Mobile robot, path planning, global planner, local planner, obstacle avoidance, Framework ROS, Stage, Gazebo

BIBLIOGRAFICKÁ CITACE

DOSEDĚL, Miroslav. Návrh a realizace strategie plánovaní pohybu mobilního robotu. Brno, 2022. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/140228. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Stanislav Věchet.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že jsem bakalářskou práci na téma „Návrh a realizace strategie plánovaní pohybu mobilního robotu“ vypracoval samostatně s použitím pramenů a odborné literatury uvedených v seznamu, který tvoří přílohu této práce.

V Brně dne 20. května 2022 .....

Doseděl Miroslav

PODĚKOVÁNÍ

Rád bych poděkoval svému vedoucímu práce doc. Ing. Stanislavu Věchetovi, Ph.D. za cenné rady, jeho vedení a konzultace.

OBSAH

1 ÚVOD

Hlavním úkolem robotů je ulehčení lidské činnosti. Vykonávají monotónní činnosti, práce, při kterých je potřeba vysoká přesnost, starají se o nebezpečné úkony a celkově se snaží zefektivnit práci, pro kterou byly navrženy. Robot je široce používaný v mnoha odvětvích jako je průmysl, zemědělství a ve velkém množství dalších aplikací. Využívají se například jako autonomní vozíky ve skladech, jako kolaborativní roboty pro práci s člověkem v průmyslu a jako autonomní vozidla. Jsou hojně používány jak ve venkovních, tak ve vnitřních prostorách. V dnešní době se začaly autonomní roboty výrazně používat i pro práci v domácnostech, kde se starají například o automatické vysávání bytu, umývání podlahy nebo o sekání trávníku na zahradě. Větší varianty těchto robotů se využívají pro udržování čistoty ve velkých prostorách jako jsou kancelářské budovy, letiště, školy, či obchodní centra.

Jednou z monotónních činností, které musí člověk vykonávat, je zalévaní rostlin. Zalévání rostlin je dobře automatizované ve velkém měřítku v zemědělství, kde je zavedeno mnoho různých druhů zavlažovacích systémů. Pro rostliny, které má člověk ve svém bytě, neexistuje mnoho způsobů, jak jednoduše a automaticky zalévat pokojové rostliny. Danou problematikou se zabývá tato diplomová práce.

Aby byly autonomní roboty schopny se samostatně pohybovat v prostoru z místa A do místa B, musí být vybaveny nějakým druhem plánování pohybu. Plánování pohybu se vybírá takové řídící povely, aby se mobilní robot dostal do cílové pozice, a při tom se vyhnul všem překážkám, které se nachází v jeho cestě. Pro správné vyhýbání se překážkám, lidem a zvířatům, musí mít roboty dobré informace o svém okolí a jsou proto vybaveny řadou sensorů.

Cílem této práce je realizovat pohyb robota určeného pro provoz ve vnitřních prostorách budov pro zalévání pokojových rostlin. Jedná se především o návrh pohybu robotu jak v prostoru, kde se robot pohybuje mezi jednotlivými rostlinami i zdroji vody, tak v blízkosti rostliny, kterou má zalít. Dále práce navrhuje metodu pro navigaci robotu do požadované zalévací pozice, ze které bude robot schopen rostliny zalít. Závěrem integruje pohyby do řídicího systému robotu a následně hodnotí dané funkcionality.

Projekt zalévacího robotu je rozdělen do více diplomových prací, mezi další práce patří „Komplexní návrh zalévacího mobilního robotu“ od Ondřeje Podolinského [1], „Detekce a klasifikace objektů zájmu zalévacího robotu zpracováním obrazu“ od Jiřího Sladkého [2], „Návrh a realizace komunikačního rozhraní autonomního mobilního robotu“ od Jana Bajera [3] a „Návrh zalévacího modulu pro mobilní robot“ od Petra Vizváryho [4].

2 MOTIVACE A ROZBOR PRÁCE

Práci lze rozdělit do několika hlavních částí. V první části jsou představeny algoritmy používané pro plánování cesty v prostoru a vyhýbání se překážkám. V další části je představen Framework ROS použitý pro ovládání robotu a práci se sensory.

Ve druhé části je představen robot, který byl předělán na zalévacího robota připevněním zalévací platformy. Dále je uveden princip lokalizace robotu v prostoru. Pro implementaci a odzkoušení algoritmů plánování pohybu bylo využito simulačního prostředí kompatibilního s frameworkem ROS. Simulace byla vytvořena v programu Stage a následně v programu Gazebo, z důvodu nedostupnosti robotu. Simulace v programu Gazebo, byla ze začátku náročná na výkon počítače, proto se pro otestování algoritmů nejdříve používal jednodušší program Stage. Po optimalizaci simulace v programu Gazebo se přešlo k tomuto simulačnímu prostředí.

Ve třetí části je uvedeno rozdělení pohybů pro jednotlivé činnosti robotu a následně jejich detailní představení a otestování v simulačním prostředí.

Ve čtvrté části jsou uvedeny druhy komunikace, které probíhají mezi aplikací a robotem. Robot předává aplikaci informace o své poloze a o prostředí, ve kterém se pohybuje. Tato data jsou využita k vizualizaci okolí robotu v aplikaci.

V páté části je popsáno a zhodnoceno testování plánování pohybu na reálném robotu.

3 REŠERŠE

V následující kapitole budou nejprve prozkoumány existující algoritmy pro řešení problému plánování pohybu v prostoru. V další části je rozebrán Framework ROS a jeho použití.

3.1 Plánování pohybu

V této podkapitole bude popsán problém plánování pohybu. Při zpracování se vycházelo hlavně z [5] a [6].

Plánování pohybu se zabývá nalezením cesty, po které se robot pohybuje ze startovní pozice do cílové a vyhýbá se překážkám. Problém plánování pohybu lze uvést takto: je dána známá pozice robotu, požadovaná pozice cíle, geometrický popis robotu a geometrický popis prostředí robotu. [7]

3.1.1 Pracovní prostor robotu

Pracovní prostor je prostor, ve kterém se robot pohybuje. Pracovní prostor obsahuje překážky, které lze rozdělit na statické a dynamické. Statické překážky se nemění v čase, například zdi. Překážky dynamické mění časem svoji polohu či velikost, příkladem je pohybující se člověk. V prostoru se může robot nacházet v určité konfiguraci, která popisuje polohu a natočení robotu pomocí vektoru souřadnic q v konfiguračním prostoru. Konfigurační prostor C (C-Space) je prostor všech možných konfigurací. Skládá se z volné oblasti konfigurací C_{free}, kde nedochází ke kontaktu robotu s překážkami, a C_{obs}, kolizní oblasti konfigurací, kde je robot v kontaktu s překážkou. [7]

3.1.2 Vymezení základních pojmů

Algoritmy pro hledání cesty v prostoru se rozdělují podle následujících kritérií. Dělí se na plánovače pohybu ve vnitřním nebo venkovním prostředí, plánovače pohybu s časovou nebo bez časové informace, plánovače pracující v reálném čase nebo plánovače trasy před pohybem robotu. Může se hledat cesta, která je nejkratší, cesta s nejlepším ohodnocením, cesta, která trvá nejkratší dobu, nebo pouze bezkolizní trasa. Dále se rozlišuje, jestli robot dojede přibližně do cílové polohy, nebo je potřeba přesného pozicování. [7]

Plánování pohybu se nejčastěji převádí na prohledávání grafu. Graf se skládá z uzlů propojených hranami. Podle toho, jestli lze hrany následovat jakýmkoliv směrem, se graf rozděluje na orientovaný nebo neorientovaný. Jestliže graf obsahuje cenu přechodu mezi uzly neboli váhu, jedná se o vážený graf.

Speciálním typem orientovaného grafu je strom. Strom má jeden začáteční uzel a ostatní uzly mají vždy jen jednoho rodiče. Uzly mohou být dosaženy pouze přes jednu hranu z jediného jiného uzle. Každý uzel bez potomka je nazýván list. Strom může být vážený.

3.1.3 Metody plánování cesty robotu

Existuje několik metod pro plánování cesty robotu. Většinou jsou složeny ze dvou částí. Nejdříve je diskretizovaný pracovní prostor převeden do konfiguračního prostoru. Volný prostor se reprezentuje jako graf, kde uzly představují konfigurace robotu a hrany volné cesty mezi konfiguracemi. Jakmile máme sestrojený graf, můžeme najít cestu mezi počáteční a koncovou konfigurací pomocí prohledávání grafu. Tyto metody jsou popsány v následujících podkapitolách.

3.1.4 Metoda rozkladu do mřížky

Metoda rozkladu do mřížky je založena na rozdělení prostředí, ve kterém se pohybuje robot, do buněk mřížky. Každá buňka je následně reprezentována v grafu. [8]

Existují dva druhy dekompozice – přesná a přibližná.

Při přesné dekompozici (Exact Cell Decomposition) je volný prostor rozložen, za pomoci přímek směřujících nahoru a dolů z každého vrcholu, do různoběžníků se svislými bočními segmenty. Grafová struktura je tvořena umístěním jednoho bodu do středu každého polygonu a do každého vertikálního segmentu. Na konec jsou tyto body spolu propojeny. [9]

Přibližná metoda rozkladu (Approximate Cell Decomposition) používá buňky se stejným předem definovaným jednoduchým tvarem. Nejčastěji se jedná o buňky čtvercového tvaru. Pokud obsahuje buňka alespoň kousek překážky, označí se buňka jako obsazená. V opačném případě se označí jako volná. Výhodou oproti přesné dekompozici je jednodušší implementace a numerická stabilita. Velikost buněk ovlivňuje jak přesnost řešení, tak výpočetní náročnost. S vyšším počtem buněk je nalezené řešení přesnější, na druhou stranu roste s počtem buněk i výpočetní náročnost. [9]

3.1.5 Metoda mapy cesty

Speciálním typem grafu je mapa cesty (Roadmap), která reprezentuje volný pracovní prostor. Mapa cesty zaručuje, že pokud cesta existuje, plánovač ji najde. Aby byl graf mapou cesty, musí splňovat dvě podmínky. Musí existovat propojení mezi mapou cesty pro každou konfiguraci z volného prostoru a musí být možné nalézt cestu mezi jakoukoliv konfigurací q a některým z uzlů mapy cesty. Pokud jsou tyto podmínky splněny, pak lze pomocí mapy cesty nalézt řešení. Metody založené na mapě cesty se dají rozdělit na deterministické a pravděpodobnostní. [10]

Deterministické metody

Jen zřídka se vytváří mapa cesty, která plně popisuje celý pracovní prostor, protože se jedná o obtížnou úlohu. Existuje pouze několik případů, kdy je to jednoduché. Pro následující případy předpokládejme robota a prostředí vytvořené pouze z polygonů. Typickými zástupci deterministické metody jsou graf viditelnosti a Voronoiův diagram. [10]

Graf viditelnosti

Prvním krokem při tvorbě grafu viditelnosti je transformace překážek do konfiguračního prostoru. Jelikož jsou jak překážky, tak konfigurační prostor v rovině, je transformace jednoduchá. Robot se pouze posune kolem hran překážek a zaznamená se cesta vytvořená jeho referenčním bodem. Pokud se referenční bod nachází mimo vytyčené oblasti, leží ve volném prostoru. Sestavený graf obsahuje uzly, které představují rohy překážek v prostoru. Jsou propojeny mezi sebou pouze uzly, které lze spojit bez protnutí překážky. Pro dokončení grafu stačí propojit cílovou a počáteční pozici se všemi viditelnými uzly. Nyní lze použít algoritmus pro hledání cesty v grafu (například A*), který slouží k nalezení cesty mezi koncovou a počáteční polohou.

Tento algoritmus je konečný (vytváří mapu cesty) a optimální (nalezne nejkratší cestu). Nevýhodou je, že nalezená cesta vede v těsné blízkosti kolem překážek. Pro oddálení cesty od překážek lze překážky na začátku algoritmu lehce zvětšit. [10]

Voronoiův diagram

Jedná se o geometrickou strukturu v rovině, která je tvořena body se stejnou maximální vzdáleností od všech nejbližších překážek. Body tvoří hrany grafu a uzly jsou tvořeny body, které jsou stejně vzdáleny od tří anebo více překážek. Stejně jako v předchozím případě je potřeba počáteční a cílovou pozici napojit na nejbližší hranu grafu. [5]

Pravděpodobnostní metody

Je obtížné analyticky vytvořit úplnou mapu cesty, proto se používají různé metody pro její přibližné vytvoření. Oproti metodám založených na rozkladu prostoru do mřížky jsou pravděpodobnostní metody méně časově a výpočetně náročné, jelikož neprochází prostředí buňku po buňce. [5]

Probabilistic Roadmap – PRM

Mezi pravděpodobnostní metody patří algoritmus Probabilistic Roadmap neboli PRM. Jedná se o algoritmus založený na náhodném vzorkování pracovního prostoru. Na rozdíl od reprezentace prostoru jako mřížky má výsledný graf méně uzlů a je rychlejší. PRM se používá pro řešení komplexních problémů plánování trajektorie ve více dimenzionálním prostoru.

PRM se skládá ze dvou hlavních částí. Nejdříve se zvolí určitý počet vzorků z pracovního prostoru. Vzorky jsou vybrány náhodným rovnoměrným vzorkováním prostoru. Jsou zachovány pouze vzorky, které se nacházejí mimo překážky. Vzorky mohou být vybrány i za pomoci nerovnoměrného vzorkování pro zvýšení pravděpodobnosti, že PRM bude reprezentovat úzké chodby i s nižším počtem vzorků. Vytvořené vzorky se stávají uzly grafu. V další části jsou uzly pospojovány mezi sebou. Pro každý uzel se nalezne několik sousedních uzlů a pokusí se o nalezení cesty mezi všemi sousedními uzly a aktuálním uzlem. Pro naplánování cesty lze použít jednoduchý a rychlý lokální plánovač. Příkladem je plánovač, který propojí body přímou čarou. Pokud je tato cesta bezkolizní, přidá se hrana mezi tyto dva uzly. Toto se opakuje pro všechny uzly. Na závěr vytvořený graf přibližně odpovídá volnému prostoru (s vyšším počtem vzorků bude odpovídat přesněji a bude menší šance, že algoritmus nebude schopen najít cestu, pokud existuje). Cesta je následně získána použitím Dijkstrova algoritmu nebo algoritmu A* na výsledný graf.

Díky možnosti změnit lokální plánovač lze snadno upravit algoritmus, aby odpovídal potřebnému využití. Vytvořený graf lze použít pro nalezení cesty mezi různými pozicemi. Při použití v dynamickém prostředí je potřeba testovat hrany na jejich průchodnost či opakovaně vytvářet nový graf. [11]

Rapidly-exploring random trees – RRT

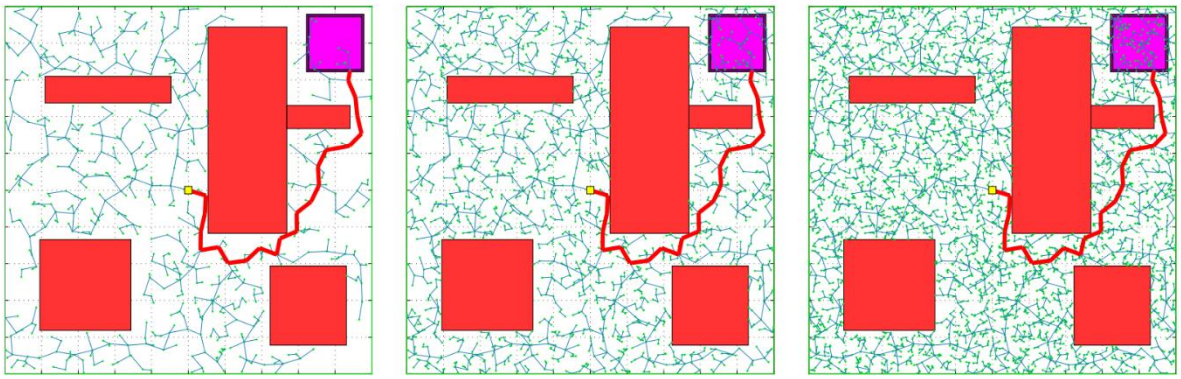
Jedná se o metodu založenou na náhodném prohledávání konfiguračního prostoru. Algoritmus Rapidly-exploring random trees neboli RRT je speciálně navržen tak, aby dokázal pracovat s neholonomním omezením a s roboty s vysokými stupni volnosti. RRT je vhodný pro prohledávání velkých zmapovaných prostředí, která by v reprezentaci v mřížce obsahovala miliony buněk.

Algoritmus funguje na základě náhodného vybírání vzorků z prohledávaného prostoru. Po vybrání vzorku je zapotřebí nalézt jeho nejbližší uzel, který je součástí stromu. Pokud se vybraný vzorek nachází dále, než je maximální vzdálenost, přesune se vzorek blíže a propojí se s již existujícím uzlem. Toto se opakuje, dokud není vytvořen uzel v blízkém okolí cílové pozice. Cesta je rekonstruována následováním sekvence rodičů zpátky z cílového uzlu.

Pro nasměrování algoritmu směrem k cíli lze v určitých časových intervalech vybrat vzorek z cílové oblasti a pokusit se ho připojit ke stromu.

Lokální plánovač může jednoduše propojit dva body přímkou nebo složitějšími tvary. Pro mobilního robota propojuje body oblouky, které představující zatáčení robotu.

RRT jsou používány pro rychlé vyřešení složitého plánování pohybu bez ohledu na kvalitu řešení. Vhodné je použití pro nalezení cesty k cíli a zjištění, zda cesta k němu opravdu existuje. [9] [12] [6]

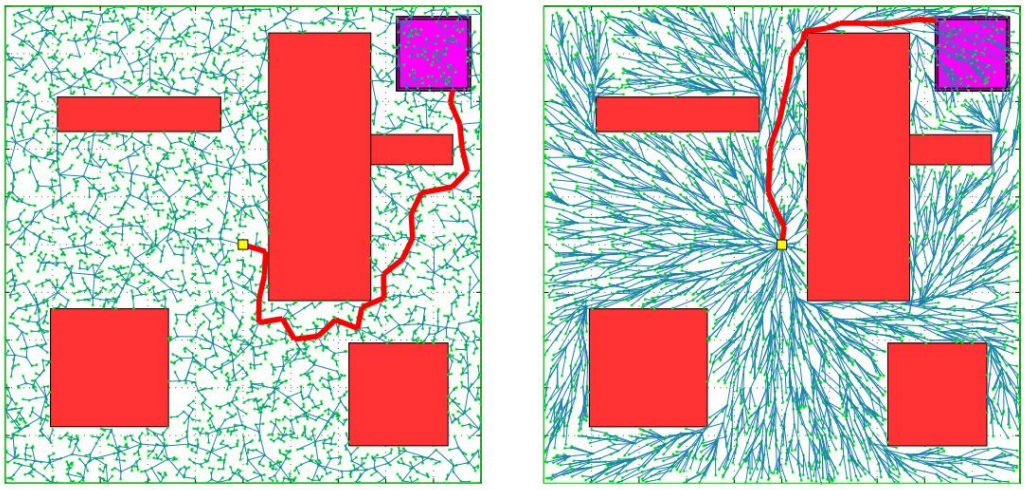


Obrázek 3-A – Porovnání algoritmu RRT s různým počtem iterací 1000, 2500 a 5000 iterací, převzato z [21]

RRT*

Pro nalezení lepšího řešení existuje modifikace s názvem RRT*. Funguje stejným způsobem jako RRT. Liší se místem, kde se připojí nový uzel. Ne vždy se jedná o nejbližší uzel. Prozkoumá se okolí uzlu a vybere se takový uzel, aby byl zachován strom a zároveň byla minimalizovaná celková trasa k uzlu. Algoritmus RRT* neustále mění tvar výsledného stromu, takže nalezené řešení se s rostoucím počtem uzlů blíží optimálnímu řešení.

Výsledný strom dokáže propojit počáteční pozici s dalšími body prostředí a vytváří tak optimální trasu k jakémukoliv bodu v prostoru za podmínky, že prostředí bude neměnné. RRT* se nejčastěji používá pro plánování pohybu ještě před jeho začátkem. Nevýhodou tohoto algoritmu je nemožnost použití pro libovolný druh robotu. Na druhou stranu je jeho výhodou jednoduchost a výkonnost u složitějších problémů plánování pohybu. [6]



Obrázek 3-B – Vizuální porovnání RRT a RRT* pro 2500 iterací, převzato z [21]

3.1.6 Metoda potenciálových polí

Poslední metodou reprezentace pracovního prostoru je metoda potenciálových polí. Robot a překážky se chovají jako kladně nabité částice a cíl jako záporně nabitá částice. Přičemž překážky odpuzují robota tím, že vytvářejí odpudivou sílu. Cíl robota naopak přitahuje díky opačnému náboji. Výsledná síla působící na robota je součet všech odpudivých a přitažlivých sil:

$$F(q) = F_{goal}(q) + \sum_{i} F_{obs_i}(q) \tag{1}$$

, kde $F(q)$ je celková síla působící na robot, $F_{goal}(q)$ je přitažlivá síla od cílové pozice a $F_{obs}(q)$ je odpudivá síla od překážek.

Velikost síly působící na robota je závislá na vzdálenosti. Gradient je záporný z důvodu, že pokud se robot nachází dál od cíle, působí na něj menší síla, a čím blíže se bude blížit cíli, tím vyšší síla na něj působí.

Přitažlivá síla:

$$P = \frac{1}{2} \cdot \left(q - q_{goal}\right)^T \cdot K_{attr}(q - q_{goal}) \tag{2}$$

$$F = -\frac{\partial P}{\partial q} \tag{3}$$

$$F_{goal}(q) = -\frac{\partial P_{goal}}{\partial q} = K_{attr}(q - q_{goal}) \tag{4}$$

, kde P je potenciál, $(q - q_{goal})$ je rozdíl aktuální polohy robotu a cíle, K_{attr} je škálování, F_{goal} je přitažlivá síla a q je pozice robotu.

Odpudivá síla:

$$P_{obs}(q) = \frac{K_{rep}}{2d^2(q, B)} \tag{5}$$

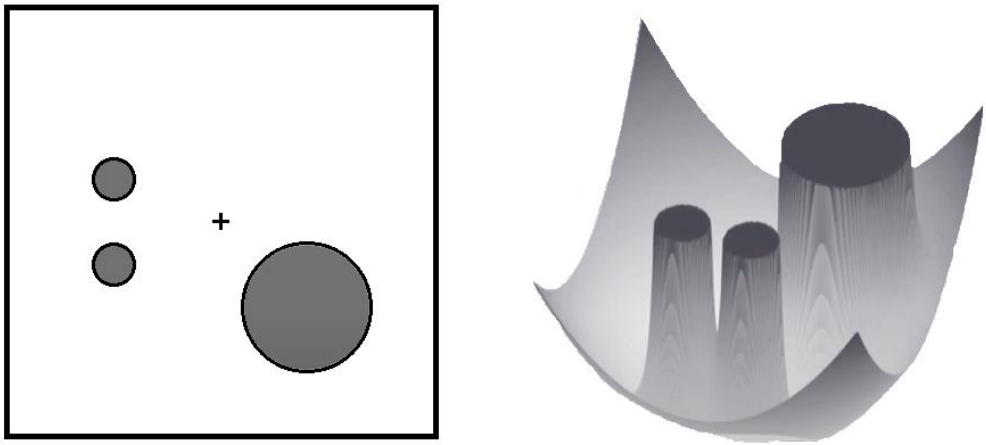
$$F_{obs} = -\frac{\partial P_{obs}}{\partial q} = \frac{K_{obs}}{d^3(q, B)} \cdot \frac{\partial d}{\partial q} \tag{6}$$

, kde $d(q, B)$ je vzdálenost od překážky, $P_{obs}(q)$ je potenciál od překážek, K_{rep} je škálování a F_{obs} je odpudivá síla od překážky.

Rychlosti robotu odpovídají síle na něj působící:

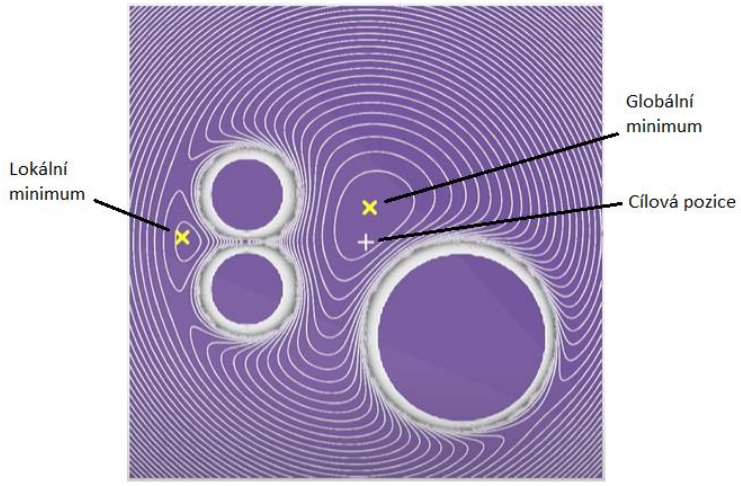
$$\dot{q} = F(q) \tag{7}$$

, kde $\dot{q}$ jsou rychlosti robotu a $F(q)$ je celková působící síla.



Obrázek 3-C – Znázornění metody potenciálových polí, pracovní prostor (vlevo) a jeho zobrazení jako metody potenciálových polí (vpravo), převzato a upraveno z [9]

Tento algoritmus vybere nejkratší cestu do cíle, ale má problém s lokálním minimem, do kterého může spadnout. Toto nastává například u symetrických překážek nebo překážek tvaru U a H. Další nevýhodou je případ, kdy globální minimum bude jiné, než je cílová pozice. Pro správnou funkčnost je potřebná přesná reprezentace prostoru, což může být až příliš nákladné. [13]



Obrázek 3-D – Znázornění možných problémů, místa s lokálním minimem a rozdílné umístění globálního minima a cílové pozice, převzato a upraveno z [36]

3.2 Prohledávání grafu

Po převedení konfiguračního prostoru do grafu, kde uzly reprezentují konfigurace robotu a hrany bezkolizní cestu mezi nimi, je potřeba tento graf prohledat a nalézt cestu mezi počáteční a koncovou konfigurací robotu.

Pro prohledávání grafu lze použít algoritmy jako je Dijkstův algoritmus nebo algoritmus A*. Tyto algoritmy lze podle znalosti pracovního prostoru rozdělit na informované a neinformované.

Neinformované algoritmy nemají žádnou informaci o prostoru, musí tedy systematicky prohledávat všechny uzly, dokud se nedostanou k cíli. Příkladem algoritmů je prohledávání do hloubky nebo prohledávání do šířky.

Naproti tomu informované algoritmy znají prohledávaný prostor a díky tomu dokáží odhadnout, jak moc se liší aktuální řešení od cílového řešení. K odhadování slouží tzv. heuristická funkce $h(x)$. Příkladem informovaných algoritmů je greedy best-first search, A* a jeho varianty, D*.

Pro ohodnocení jednotlivých uzlů se používá hodnotící funkce $f(x)$:

$$f(x) = g(x) + h(x) \tag{8}$$

, kde $g(x)$ je cena přechodu do aktuální pozice od počátku a $h(x)$ je odhadovaná cena přechodu z aktuální pozice do cílové pozice [9]

Jako heuristická funkce se dá použít například manhattanská metrika h_m nebo euklidovská metrika h_e.

$$h_m = |x - x_G| + |y - y_G| \tag{9}$$

$$h_e = \sqrt{(x - x_G)^2 + (y - y_G)^2} \tag{10}$$

, kde x, y jsou souřadnice polohy a x_G, y_G jsou souřadnice cílové polohy.

Dijkstrův algoritmus

Dijkstrův algoritmus se používá se pro nalezení nejkratší cesty. Jedná se o konečný algoritmus, tedy vždy najde cestu do cílové pozice. Jedná se o neinformovaný algoritmus, který nezná, jak se jeho nalezené aktuální řešení liší od cílového. Z tohoto důvodu v nejhorším případě projde algoritmus všechny uzly grafu. Pro nalezení cesty mezi dvěma uzly, stačí algoritmus zastavit, jakmile je nalezena nejkratší cesta k cíli.

Algoritmus potřebuje dva listy, jeden s navštívenými místy a druhý s místy, která jsou na řadě. Dále jsou potřeba vzdálenosti mezi jednotlivými uzly a rodiče jednotlivých prozkoumaných uzlů. Algoritmus začíná v počátečním uzlu a všechny vzdálenosti jsou nastaveny na nekonečno. Z počátečního uzlu se vypočítá vzdálenost se sousedícími uzly. Pokud je kratší než přiřazená vzdálenost, je nahrazena novou vzdáleností a přiřadí se mu jako rodič počáteční uzel. Z nenavštívených uzlů se vybere další uzel, který se nastaví jako aktuální, a přesune se z nenavštívených mezi navštívené uzly. Toto se opakuje, dokud není nalezena koncová pozice, nebo nebyly prozkoumány všechny uzly. [9] [14]

A*

Jedním z nejpoužívanějších algoritmů je algoritmus A*. Stejně jako Dijkstrův algoritmus, se A* používá na nalezení nejkratší cesty mezi dvěma body. Na rozdíl od Dijkstrova algoritmu je A* rychlejší díky používání heuristiky při rozhodování, do jakého následujícího uzlu se má posunout při hledání cesty. A* hledá pouze nejkratší cestu mezi počátečním a koncovým uzlem. Z důvodu, že se pro ohodnocení uzlů používá heuristická funkce, může dojít k případu, kdy nalezená cesta nebude ve skutečnosti nejkratší. Aby se to nestávalo, je zavedený předpoklad, že

$$h(n) \leq h^*(n) \tag{11}$$

, kde $h^*(n)$ je reálná cena a $h(n)$ je odhadovaná heuristika.

Pro svoji činnost používá algoritmus dva listy – jeden s uzly, které již byly prozkoumány, a druhý s uzly, které budou teprve prozkoumány. Ve zkratce funguje algoritmus takto:

Nejdříve se vybere uzel s nejnižším ohodnocením (nejnižší hodnotu funkce f). Pro každý sousední uzel sousedící s vybraným uzlem se vypočítá cena cesty mezi těmito dvěma uzly. Pokud je cena sousedního

uzlu nižší, než jeho aktuální cena, je přidán uzel do seznamu s uzly, které budou teprve prozkoumány. Aktuální uzel se přesune mezi uzly, který jsou prozkoumány.

Nevýhodou použití algoritmu A* na mřížce je, že nejkratší cesta se většinou drží v blízkosti překážek. Vytvořená trasa je zarovnaná v mřížce, a pokud se při pohybu zjistí, že je v cestě překážka, je nutné celou cestu přeplánovat.

Aby se cesta držela dál od překážek, je možné použít na mapu konvoluci s Gaussovým jádrem. Překážky se pak budou jevit větší, než ve skutečnosti jsou. Hodnotící funkce se následně změní.

$$f(n) = g(n) \cdot p_{occ}(n) + h(n) \tag{12}$$

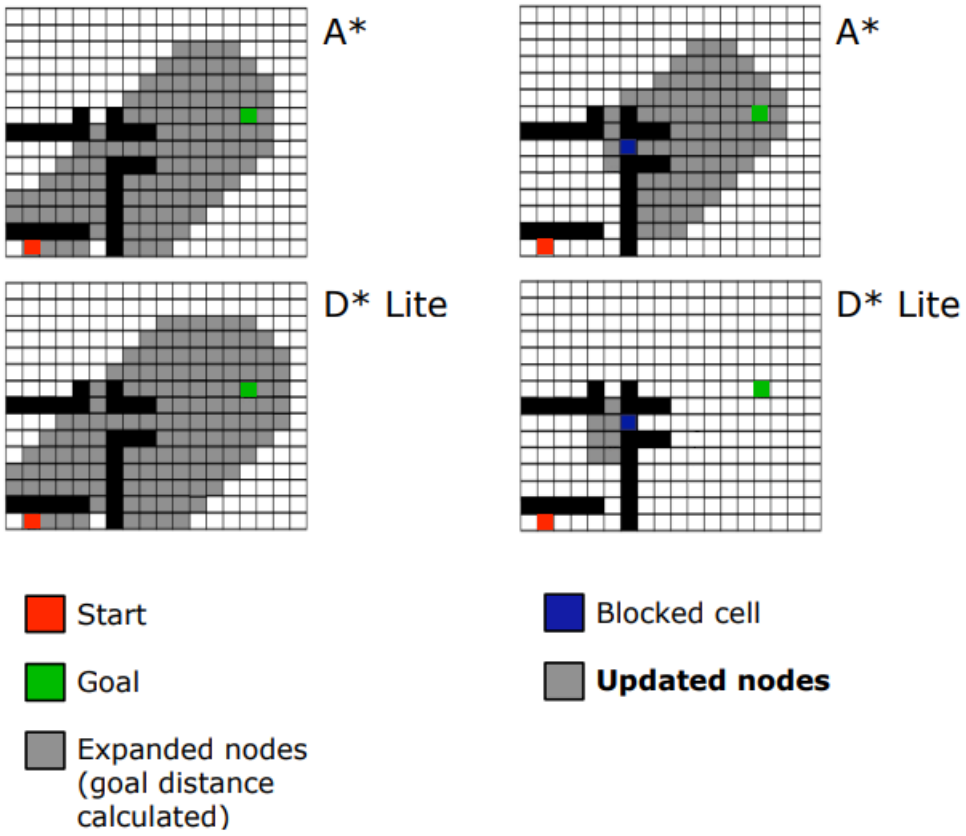
, kde $p_{occ}(n)$ je pravděpodobnost výskytu překážky v buňce. [15] [9]

D* / D Lite

Existuje mnoho dalších variací, které jsou založeny na principu algoritmu A*. Mezi ně patří D* nebo D* Lite. Jejich název vychází ze zkratky „Dynamic A*“. Tyto varianty jsou vhodnější pro dynamické prostředí. Pokud bychom v tomto případě použili algoritmus A*, musel by v pokaždé, když je cesta zablokována, přepočítat celou trasu. Efektivnější by bylo přepočítat pouze část trasy, která je zablokována, toho využívá algoritmus typu D*.

Algoritmus D* na rozdíl od algoritmu A* začíná plánovat cestu z cílové pozice. Při detekci překážky je potřeba přepočítat pouze pár uzlů v blízkosti cílové pozice. Algoritmus D* je tedy schopný stavět na předchozím řešení.

Algoritmus D* Lite implementuje stejné chování jako D*, ale jak název napovídá, je jednodušší na pochopení a lze ho napsat na méně řádků. Algoritmus typu D* (přesněji Field D*) byl použit u Mars roveru Spirit a Oportunity. [6] [9]



Obrázek 3-E – Porovnání algoritmu A* a D*, převzato z [22]

3.3 Vyhýbání se překážkám

V následující kapitole budou popsány algoritmy využívající se pro lokální plánování pohybu a k vyhýbání se překážkám.

Běžně se rozděluje problém plánování pohybu robotu na globální a lokální plánování.

Globální plánovač

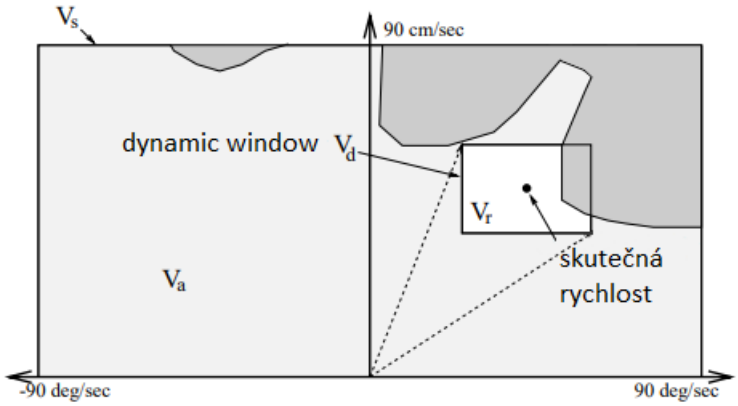
Globální plánovač se stará o naplánování cesty z místa A do místa B, často bez ohledu na kinematické a dynamické omezení robotu, k čemuž je využita mapa statických překážek. Mapa je vytvořena na základě dat ze sensorů. Globální plánovač se nejčastěji využívá pro naplánování trasy ještě před tím, než se robot rozjede. Hlavním důvodem je dlouhé plánování trasy.

Lokální plánovač

Lokální plánovač se stará o to, aby byl robot schopen se vyhýbat překážkám v blízkém okolí robotu, které nejsou zaneseny v mapě okolí, následovat naplánovanou globální trasu, při braní v úvaze omezení robotu. Výstupem z lokálního plánovače je lineární rychlost robotu v_x, v_y a úhlová rychlost ω.

Dynamic Window Aproach (DWA)

Dynamic Window Aproach neboli DWA je plánovač pohybu, který slouží k vyhýbání se překážkám v reálném čase. Vychází z dynamiky robotu a dokáže pracovat s limitací robotu jako je jeho maximální rychlost nebo zrychlení. Je vhodný pro mobilní roboty pohybující se vyššími rychlostmi. Dynamika robotu je použita při zmenšení velikosti prohledávaného prostoru pouze na takzvané *dynamické okno* (dynamic window), které se skládá z rychlostí dosažených robotem v určitém časovém intervalu. Rychlosti se zadávají robotu ve tvaru v, ω (jsou kombinací lineární a úhlové rychlosti robotu). Z těchto rychlostí je potřeba vybrat pouze ty rychlosti, které dostanou robota do cíle, jsou bezkolizní a robot jich dokáže dosáhnout. [16]



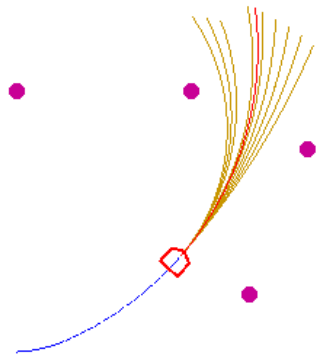
Obrázek 3-F – Zobrazení dynamického okna, převzato a upraveno z [23]

Všechny vhodné rychlosti jsou následně ohodnoceny hodnotící funkcí $G(v, \omega)$. Tato funkce se pokouší o minimalizaci jízdní doby rychlou a bezpečnou jízdou správným směrem.

$$G(v, \omega) = \alpha \cdot clearence\ (v, \omega) + \beta \cdot velocity\ (v, \omega) + \gamma \cdot dist(v, \omega) \qquad (13)$$

, kde α, β, γ jsou vážené parametry a $clearence\ (v, \omega)$ je ohodnocení vzdálenosti robotu od překážky, $velocity\ (v, \omega)$ je ohodnocení velikosti rychlosti a $dist(v, \omega)$ je ohodnocení vzdálenosti robotu od cíle.

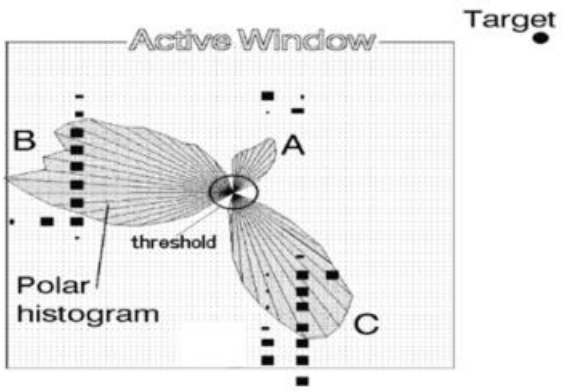
Robotu je poté zadána dvojice rychlostí s nejlepším ohodnocením. Mezi hodnoticí parametry patří vzdálenost robotu od překážky, lineární rychlost robotu či vzdálenost robotu od cílové pozice. Parametry α, β, γ jsou váhy jednotlivých parametrů. [17] [9] [18]



Obrázek 3-G – Reprezentace možných cest robotu, převzato z [32]

Vector Field Histogram – VHF

Vector Field Histogram neboli VHF je algoritmus pro vyhýbání se překážkám v reálném čase. Je založený na práci s daty ze sensorů vzdáleností. VHF využívá statistickou reprezentaci okolí robotu společně s překážkami. Na rozdíl od jiných algoritmů pro vyhýbání se překážkám bere VHF v potaz dynamiku a tvar robotu a vrací příkazy pro řízení robotu. VHF je jedním z nejpopulárnějších lokálních plánovačů používaných v mobilní robotice společně s DWA. Jedná se o robustní algoritmus, který není náchylný na chybová data ze senzorů, proto je vhodný pro použití s lidarem.



Obrázek 3-H – Zobrazení polárního histogramu, převzato z [24]

VHF algoritmus se skládá ze tří hlavních kroků:

Prvním krokem je vytvoření 2D histogramu kolem robotu reprezentující překážky. Tento histogram je aktualizován za pomoci senzorů v reálném čase. Ve druhém kroku se převede 2D histogram na 1D polární histogram, centrovaný v momentální pozici robotu. V poslední částí se zvolí nejlepší oblast s nejmenším obsahem překážek a určí se rychlosti robotu, aby se robot vydal tímto směrem.

Nevýhodou VHF je potřeba vyššího výpočetního výkonu a skutečnost, že ne vždy dokáže dostat robota do cílové pozice, protože se u něj vyskytuje problém se symetrickými překážkami tvaru písmene H nebo U. [19] [20]

Bug Algoritmus

V následující části bude popsán bug algoritmus a jeho druhy. Bug algoritmus se používá pro vyhýbání se překážkám.

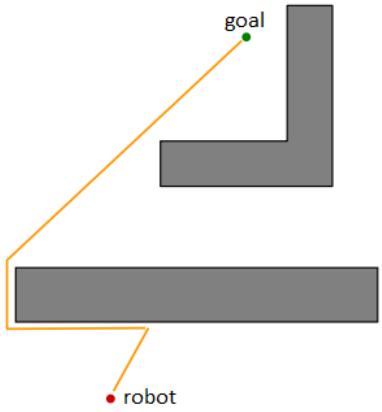
Pro Bug algoritmus není potřeba znát okolí robota, tedy překážky v prostoru mohou být neznámé. Informace o překážkách jsou získávány ze senzorů. Jak název napovídá tento algoritmus je inspirován hmyzem a snaží se nalézt cestu kolem překážek. Pokud existuje řešení, tento algoritmus ho najde a pokud řešení neexistuje, algoritmus je schopen to nahlásit. V určitých případech je schopen nalézt optimální řešení. [21]

Předpoklady:

- jedná se o dvourozměrný prostor plný neznámých překážek,
- každá překážka je jednoduchá uzavřená křivka konečné délky a nenulové tloušťky,
- překážky se navzájem nedotýkají,
- počáteční a cílová pozice je známá.

Bug 0

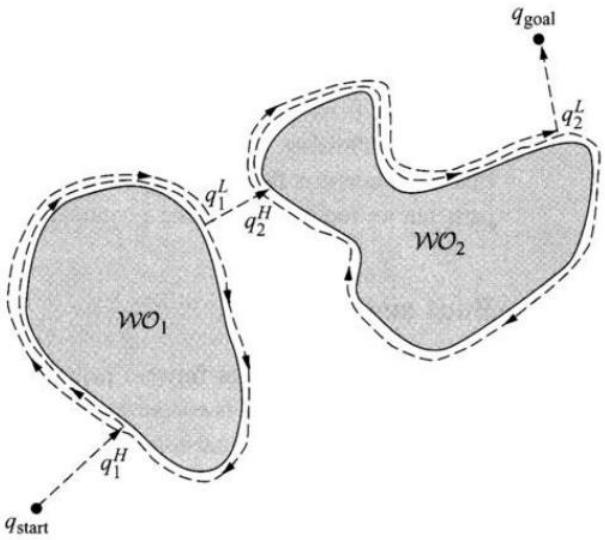
Robota si můžeme představit jako bod vybavený kontaktním sensorem, který dokáže detekovat překážku v cestě a zná svojí pozici. Robot se vydá přímou cestou k cíli, pokud sensor zaznamená překážku před sebou, začne následovat hranici objektu, dokud nemůže znovu směřovat směrem k cílové pozici. Toto se opakuje, dokud se robot nedostane do cílové pozice. Nevýhodou tohoto algoritmu je, že může nastat případ, kdy se robot nedostane do cíle, ikdyž cesta existuje. [21]



Obrázek 3-I – Princip pohybu Bug 0, převzato z [25]

Bug 1

Tento algoritmus funguje podobně jako Bug 0. Robot je v tomto případě stejný jako v předchozím typu. Na začátku se robot vydá přímo k cílové pozici. Po kontaktu sensoru s překážkou začne robot překážku

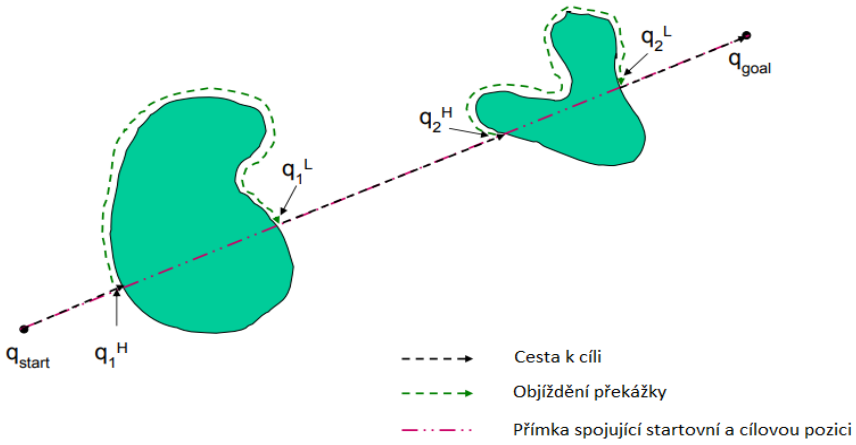


Obrázek 3-J – Princip pohybu Bug 1, převzato z [25]

objíždět. Na rozdíl od algoritmu Bug 0, robot objede celou překážku kolem dokola, a přitom si zaznamenává, jak blízko byl v každém bodě k cílové pozici. Po objetí překážky se robot vrátí na pozici, kde se nacházel nejblíže k cíli, a pokračuje přímo k němu. [21]

Bug 2

Algoritmus Bug 2 vychází z předchozího algoritmu Bug 1. Tento algoritmus funguje stejně jako Bug 1. Robot v tomto případě však neobjíždí celou překážku kolem dokola, ale pouze částečně. Na začátku algoritmu se vytvoří přímka *m* mezi počáteční pozicí a cílovou pozicí. Následně se robot rozjede po této přímce směrem k cíli. Pokud sensorem zaznamená překážku, začne ji objíždět, dokud nenarazí na místo, které protíná přímku *m*, v tomto případě začne znovu cestovat přímo do cílové pozice. [21]



Obrázek 3-K – Princip pohybu Bug 2, převzato a upraveno z [25]

Bug Tangent

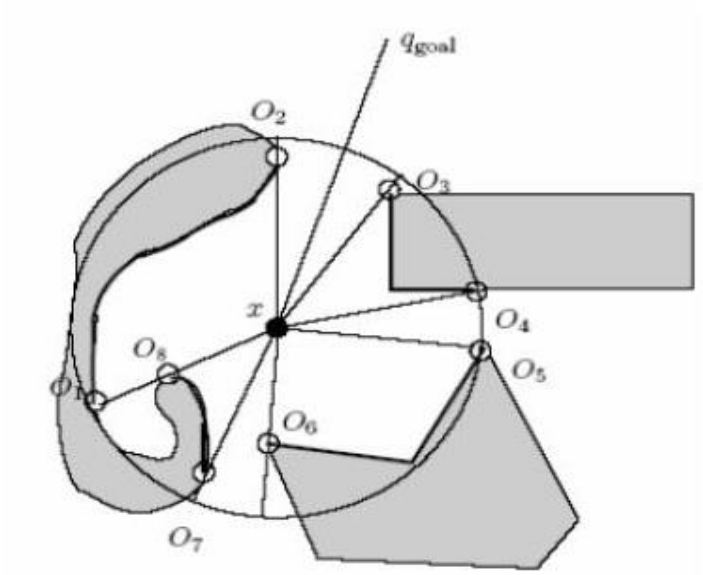
Na rozdíl od předchozích algoritmů využívá Bug Tangent sensor vzdálenosti. Pomocí tohoto sensoru zjišťuje, kde se nachází okraje spojitých překážek. Sensor vzdálenosti dává informace o okolí robotu v rozsahu 360°. Jednou z nejdůležitějších vlastností sensoru je, že je určena pouze vzdálenost k nejbližším překážkám. Sensor může být popsán tímto modelem:

$$\rho_R(x,\theta) = \begin{cases} \rho(x,\theta) & \text{pro } \rho(x,\theta) < R \\ \infty & \text{pro } \rho(x,\theta) \geq R \end{cases} \tag{14}$$

, kde $\rho_R(x,\theta)$ je vzdálenost nejbližší překážky pro určitý paprsek laseru, pod úhlem θ, R je konečný maximální dosah sensoru.

Pokud paprsek nenarazí na žádnou překážku, vrátí hodnotu jako nekonečno.

V závislosti na datech ze sensoru vzdáleností dokáže algoritmus detekovat intervaly spojitosti. Intervaly jsou zakončeny na každé straně nespojitými body. Tyto intervaly znázorňují pozice překážek a místa volného prostoru.



Obrázek 3-L – Spojité a nespojité intervaly překážek a volného prostoru, převzato z [26]

Bug Tangent využívá tyto data, díky kterým nalezne cestu do cílové pozice nebo zjistí, že je nedosažitelné. Algoritmus se skládá ze dvou hlavních částí – pohyb robotu k cíli a objíždění překážky.

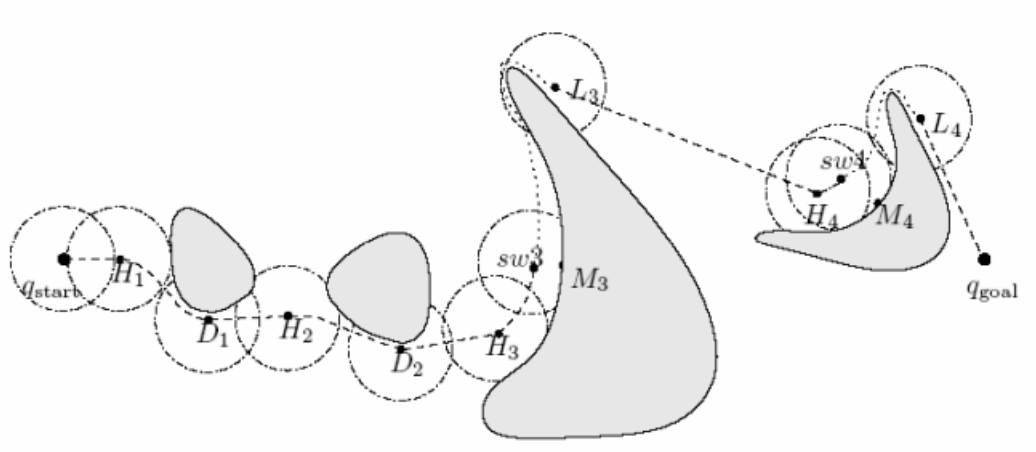
Pohybu robotu k cíli se vykonává, pokud není žádná překážka nalezena ve směru, ve kterém leží cíl nebo robot nachází snižující se hodnotu heuristické vzdálenosti od sledované překážky. Robot určí směr k cíli a začne se pohybovat tímto směrem za předpokladu, že tímto směrem nejsou detekovány žádné překážky (sensor vrací jako hodnotu nekonečno). Při nalezení překážky ve směru k cíli detekuje robot místa nespojitosti na dané překážce a vypočítá heuristickou funkci.

$$h = d(x, O_i) + d(O_i, q_{goal}) \tag{15}$$

, kde $d(x, O_i)$ je vzdálenost mezi robotem a místem nespojitosti a $d(O_i, q_{goal})$ je vzdálenost mezi místem nespojitosti a cílovou polohou.

Robot nalezne místa, kde se heuristická funkce snižuje a vydá se tím směrem. Když v jednu chvíli nastane situace, kdy se heuristická funkce nikde nesnižuje, začne robot překážku objíždět.

Při objíždění překážky se určí dva hlavní parametry: d_{reach}, který zahrnuje nejkratší vzdálenost dosahu robota (body na překážce a sensoru robota) a $d_{folowed}$, nejkratší vzdálenost mezi bodem na sledované překážce a cílem. Robot začne objíždět překážku směrem k poslednímu zvolenému bodu nespojitosti z předchozího kroku. Pohybuje se po tečně, která spojuje robota s nejbližším bodem na sledované překážce. Překážku následuje do té doby, dokud platí $d_{folowed} \geq d_{reach}$, v opačném případě se robot znovu začne pohybovat k cíli. [21] [22]



Obrázek 3-M – Princip pohybu Bug Tangent, převzato z [25]

3.4 Framework ROS

Při zpracování této kapitoly se vycházelo z práce [23] a [24]

Robotic Operation System neboli ROS je systém s otevřeným zdrojovým kódem (open-source), který usnadňuje vývoj robotických aplikací. Ačkoliv název napovídá, že se jedná o operační systém, není tomu tak. Jedná se spíše o strukturu, která slouží jako podpora při vyvíjení softwaru pro roboty. Poskytuje služby jako je řízení nízko úrovňových zařízení, předávání zpráv mezi procesy a správu balíčků (soubor skriptů, který řeší určitou funkcionalitu). Díky velkému zastoupení komunity využívající ROS, existuje mnoho vytvořených open-source balíčků a fóra, na kterých je zveřejněno mnoho užitečných informací pro práci s ROSem. ROS je primárně určen pro operační systémy typu Ubuntu a Mac OS, ale je možné ho použít i na operačních systémech Windows, avšak v tomto případě se jedná o experimentální verzi a není oficiálně podporovaná. [25] [26]

Obrázek 3-N – Logo Frameworku ROS, převzato z [5]

Nová verze ROSu se stabilními funkcemi je vydávána téměř každý rok. Nejnovější distribuce se nazývá Noetic. Balíčky z různých verzí (distribucí) nejsou většinu času mezi sebou kompatibilní, proto je dobré dávat pozor, pro jakou verzi ROSu je balíček určen a jestli je možné ho bez problému použít. [27]

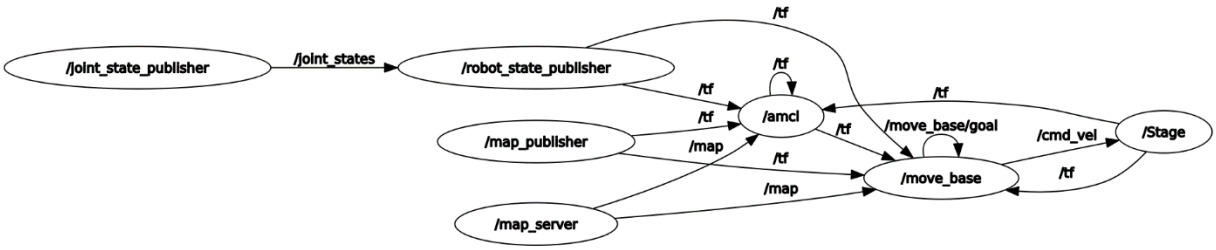
Obrázek 3-O – Distribuce ROSu, nejnovější verze Noetic (vlevo) a předcházející verze Melodic (vpravo), převzato z [7]

Celkový systém vytvořený v ROSu se skládá z mnoha dílčích částí (nodů) a každá tato část ovládá jednu nebo více funkcionalit. Jeden node se může starat o ovládání krokového motoru, další o lokalizaci robotu v prostoru a jiný o plánování pohybu. Všechny nody mezi sebou komunikují pomocí odesílání topiců do systému, voláním servisů anebo přes parametr server. Výhodou používání nodů je menší náchylnost celkového systému na chyby, jelikož jsou chyby izolovány v jednotlivých nodech. Komplexita systému je zmenšena díky rozložení systému do více částí. [28]

Pro komunikaci a výměnu informací mezi jednotlivými částmi systému slouží topic. Jedná se o pojmenované sběrnice, jejichž příjemce a odesílatel je neznámy, a jakákoliv část systému si je může přečíst. Více nodů současně může stejné zprávy odebírat a zároveň i vysílat. [29]

Pro statické parametry a proměnné, se používá parametr server. K němu mají přístup všechny nody a dokážou si z něho přečíst potřebné informace. [30]

Běžící procesy ve Frameworku ROS lze znázornit v grafové architektuře, kde uzly představují jednotlivé procesy (nody) a hrany reprezentují zprávy posílané mezi nimi.



Obrázek 3-P – Grafová reprezentace probíhajících procesů a jejich komunikace

Data neboli zprávy (messages), které se pohybují mezi nody, mají konkrétní a specifikovaný formát a strukturu. Mezi standardní typy patří int, float, bool, string atd. Pro použití vlastních datových typů je potřeba definovat vlastní typ zprávy. [31]

Existují tři hlavní způsoby, jak jednotlivé procesy mezi sebou komunikují. Jedná se o publisher-subscriber, servis a akci.

Subsrcriber a Publisher

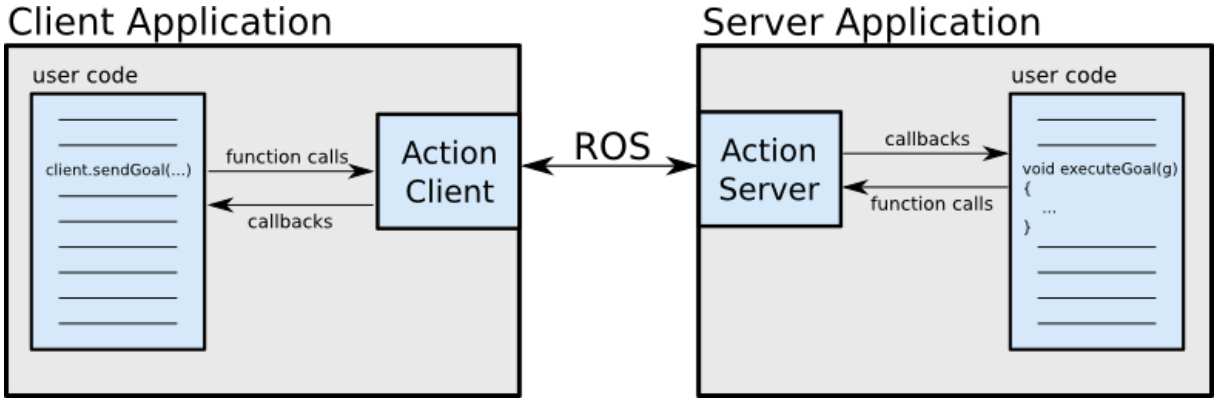
Pro vysílání zpráv z nodů se používá subscriber, který odesílá zprávy do systému. Pro jejich přijímání se naopak používá publisher.

Servis

Servis je další způsob, jakým mezi sebou nody komunikují. Na rozdíl od topiců, servisy umožňují poslat žádost o provedení činnosti jinému nodu a přijmout odpověď o jejím dokončení.

Akce

Akce se používá pro případy, kdy je potřeba provést určitou činnost, jako je spuštění plánování pohybu do určité pozice. V případě použití servisu by se začala vykonávat daná činnost a po jejím dokončení by přišla zpráva o tom, že byla dokončena. V případě déle trvající činnosti by bylo dobré mít zpětnou vazbu o jejím průběhu nebo dokonce možnost ji v některých případech zrušit. Všechny tyto vlastnosti zahrnuje akce, která je součástí balíčku *actionlib*. Akce se skládá ze dvou částí, z klientu, který slouží pro vyvolání činnosti, a ze serveru, který se stará o vykonávání dané činnosti, posílá zpětnou vazbu a informaci o jejím vykonání. [32]



Obrázek 3-Q – Zobrazení principu komunikace akcí mezi sebou, převzato z [27]

Balíčky

Balíčky jsou seskupením souborů a skriptů, které se starají o provádění určité funkcionality. Může se jednat o lokalizaci robotu v prostoru či plánování pohybu robotu. Pro Framework ROS existuje mnoho balíčků již implementovaných v sytému ROS nebo balíčků, které byly vytvořeny uživateli a jsou volně přístupné pro jejich použití. [26]

Balíčky, které jsou vhodné pro využití u této práce, jsou *amcl* pro lokalizaci robotu, a *move_base*, pro plánování pohybu robotu. Přesnější funkcionalita těchto balíčků je podrobněji popsána v následujících odstavcích.

AMCL

Pro lokalizaci robotu slouží balíček *amcl*. Jedná se o pravděpodobnostní lokalizační systém pro robota pohybujícího se ve 2D prostoru. Implementuje adaptivní lokalizační přístup Monte Carlo, který používá částicový filtr ke sledování pozice robotu na známé mapě. *Amcl* používá data ze senzorů pro určení pozice robotu ve známém prostoru. [33]

Move_base

Balíček *move_base* slouží k naplánování a vykonání cesty z počátečního do cílového bodu ve 2D prostoru a určuje rychlosti, které zabezpečí, aby robot následoval cestu a vyhýbal se překážkám. Balíček je složen z lokálního a globálního plánovače a využívá diskretizaci prostoru do mřížky. Globální plánovač slouží k nalezení trasy, nepočítá ale s překážkami a změnou okolí robotu. Lokální plánovač se stará o dynamické změny mapy a objíždění překážek. Vstupní hodnotou tohoto balíčku je cílová pozice, mapa prostředí a data ze senzorů (odometrie, lidar atd.). Výstupem je požadovaná rychlost robotu.

Pro správnou funkčnost balíčku je potřeba nastavit parametry plánovače jako je maximální a minimální rychlost a zrychlení, parametry mapy prostředí, kam se zadává například druh sensorů, velikost robota a doporučená vzdálenost robota od překážek. [34]

4 UPŘESNĚNÍ CÍLŮ

Hlavním cílem této práce je navrhnout a realizovat plánování pohybu robotu určeného pro zalévání rostlin ve vnitřních prostorách budov.

Prvním cílem je navrhnout architekturu plánování pohybu robotu během zalévání pokojových rostlin. Jelikož se jedná o komplikovaný pohyb a robot se pohybuje jak ve volném prostoru, tak v těsné blízkosti květináče, je vhodné tento pohyb rozdělit do více dílčích částí.

Druhým a třetím cílem je navržení metody globálního plánování pohybu robotu v prostoru a navržení metody lokálního plánování pohybu v oblasti zalévání. Jedná se o část pohybu, kdy se robot přesouvá mezi rostlinami a čerpacím modulem. Při pohybu v prostoru je využíván globální i lokální plánovač.

Předposledním cílem je navržení metody tzv. mikroplánování kolem vybrané rostliny pro navigaci do požadované zalévací pozice – umožnění robotu dostat se do takové vzdálenosti ke květináči, aby byl robot schopen rostlinu zalít nebo ji objet.

Navržené algoritmy je následně potřebné integrovat do řídícího systému. Protože se integrací jednotlivých funkcí zabývá jiná diplomová práce „Komplexní návrh zalévacího mobilního robotu“ [1], budou jednotlivé algoritmy pouze připraveny pro integraci do řídícího systému.

Poslední částí je zhodnocení a ověření funkcionalit vybraného řešení.

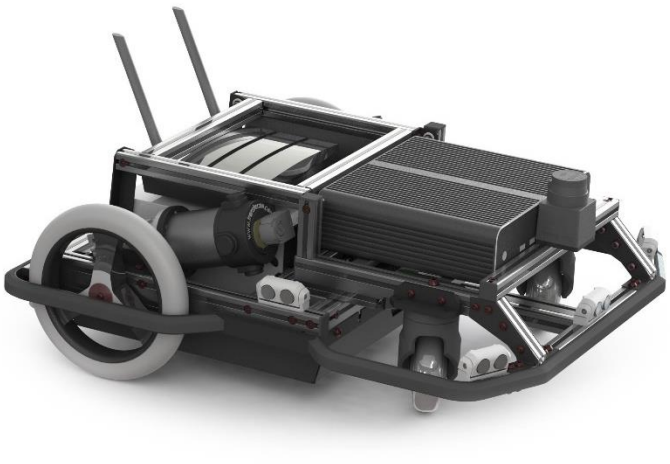
5 NÁVRH PLÁNOVÁNÍ POHYBU

V následujících podkapitolách bude popsán robot, který byl použit při realizaci zalévacího robotu a jehož model byl využit v simulaci, a způsob, jakým je prováděna lokalizace robotu v prostoru a popis použitých simulačních prostředí.

5.1 Robot Breach

Pro vytvoření mobilního robotu pro zalévání rostlin existují dvě možnosti, které lze využít. Buď vytvořit nového robota, nebo použít již existujícího robota a pouze ho poupravit. Jelikož je jednodušší poupravit existujícího robota, byla vybrána druhá varianta.

Robot použitý v této práci byl zapůjčen firmou Bender Robotics. Jedná se o robota Breach.



Obrázek 5-A – Robot Breach, který byl zapůjčený firmou Bender Robotics. Převzato z [16]

Robot Breach je určený pro použití ve vnitřních prostorech, což vyhovuje využití jako zalévacího robotu. Jedná se o mobilní robot s diferenciálním podvozkem, robot je schopen otočit se na místě a s pohonem dvou kol. Velikost robotu je přibližně 62x54x25 cm. [35]

Robot obsahuje několik sensorů, a to lidar a odometr, díky kterým je možné přesně lokalizovat robot v prostoru.

5.2 Lokalizace robotu

Lokalizace robotu v prostoru probíhá za pomocí balíčku *amcl* (Adaptive Monte Carlo Localization). Modul v tomto případě bere data ze senzorů (odometrie a lidaru) a porovnává je s dostupnou mapou. Pro správné fungování lokalizace je potřeba nastavit několik parametrů. Mezi parametry patří očekávaný šum z odometrie a počáteční odhad polohy robotu, kde se nachází. [33]

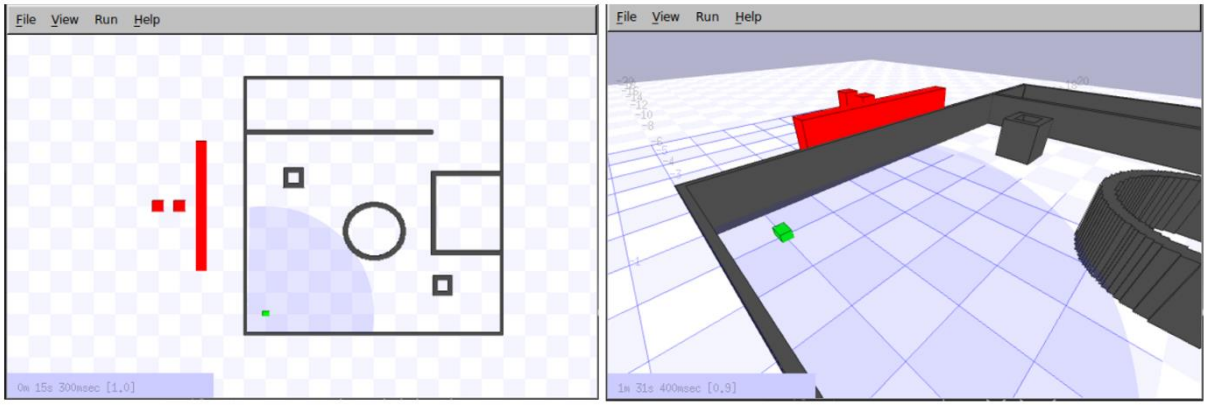
5.3 Simulační prostředí

Jelikož nebyl robot Breach ze začátku psaní práce k dispozici, probíhala většina činnosti v simulačním prostředí. Simulace probíhala ve dvou programech. Nejdříve byl pro simulaci použit program Stage, pro jednoduchou simulaci pohybu robotu v prostředí z důvodu poskytnutí vizualizace pohybu robotu, nízké náročnosti na výkon počítače a jednoduchého propojení s Frameworkem ROS a důvodu, že tvorba simulace v programu Gazebo byla součástí jiné diplomové práce. Po vytvoření simulace v programu Gazebo se přesunula simulace pohybu robotu do tohoto programu.

5.3.1 Stage

Prvotní simulace robotu probíhaly v programu Stage. Jedná se o program sloužící pro simulování robotu. Obsahuje možnosti k vytvoření virtuálního světa s roboty a modely jejich sensorů a aktuátorů zahrnujících sonar, lidar, rozpoznávání barvy, nárazníků, koncových efektorů a podvozky robotů s odometrickou nebo globální lokalizací. Jedná se o simulaci hardwaru a ROS se chová, jako by komunikoval s reálným systémem.

Softwar Stage byl využit hlavně z důvodu nízké náročnosti na výkon počítače oproti jiným softwarům pro simulaci (Gazebo). Velikou nevýhodou bylo na druhou stranu menší množství dostupných informací k práci s tímto softwarem. Stage byl využit pro otestování plánování pohybu robotu v prostoru a dále k lokalizaci robotu v prostoru. Z důvodu nutnosti komplikovanější simulace, která by se v programu Stage nedala lehce uskutečnit, se přešlo k softwaru Gazebo.



Obrázek 5-B – Simulační prostředí Stage

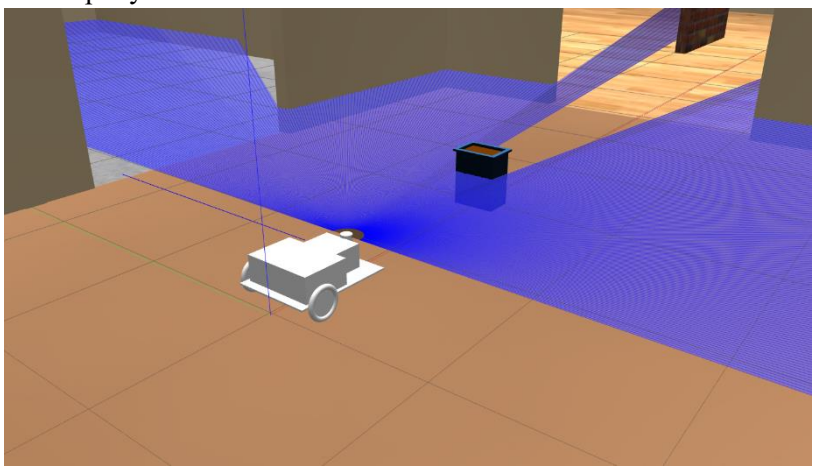
Pro nastavení simulace je potřeba vytvořit soubor s příponou .world, ve kterém je popsáno prostředí a robot. Prostředí je popsáno pomocí bitmapy, která je přizpůsobena, aby pokryla požadovaný prostor a podle zadané výšky vytvořila stěny, které je možné detekovat lidarem. Robot je tvořen jako skupina souřadnic, které popisují tvar robotu. Aby byl robot schopen detekovat překážky je potřeba vytvořit lidar a připojit ho k robotu. Důležitým parametrem je origin (počátek) souřadného systému, ke kterému je vázán daný objekt a kolem kterého jsou prováděny transformace a rotace objektu. Posledním důležitým prvkem, který je možné přidat do prostoru, jsou překážky, které lze stavět robotu do cesty a sledovat reakce plánovače pohybu na nové překážky. [36]

5.3.2 Gazebo

Druhá simulace byla vytvořena v programu Gazebo. Jedná se o open-source simulátor robotu. Na rozdíl od programu Stage zvládá Gazebo i 3D simulaci. Poskytuje možnosti pro realistickou reprezentaci

prostředí s osvětlením, stíny i texturami prostředí. Dokáže modelovat sensory jako je například lidar, či kamery. [37] [38]

Vytvoření simulace robotu v prostředí Gazebo bylo součástí jiné diplomové práce „Komplexní návrh zalévacího mobilního robotu“ [1]. Tato část je uváděna hlavně z důvodu nutnosti používání dané simulace pro plánování pohybu robotu.



Použití této simulace dovolilo vymodelovat přesnou reprezentaci robotu s jeho senzory a květináče v prostoru, ke kterým se měl robot dostat. Nevýhodou této simulace byla vyšší výpočetní náročnost, kvůli které běžela simulace velmi pomalu a bylo obtížné zaznamenat chyby, které se mohly v simulaci objevit. Jednou z těchto chyb bylo ujíždění robotu do jedné strany. Při zadání přímého pohybu robotu (zadání pouze lineární složky rychlosti v ose x) se robot nepohyboval přímým směrem, ale začal významně zatáčet na jednu stranu. Tato skutečnost nebyla po delší dobu nijak zpozorována, protože testovaný lokální plánovač byl celkem schopen si poradit s touto skutečností. Jediným viditelným projevem bylo robotovo občasné vyrovnávání natočení při přímočarém pohybu a občasným nárazem do zdi, pokud jel robot v její blízkosti a ujížděl směrem k ní. Po dlouhé době strávené nad hledáním chyby v plánování pohybu, byla tato chyba nalezena v simulaci. Chyba byla vzápětí opravena společně s optimalizací rychlosti simulace na přijatelnější hodnoty.

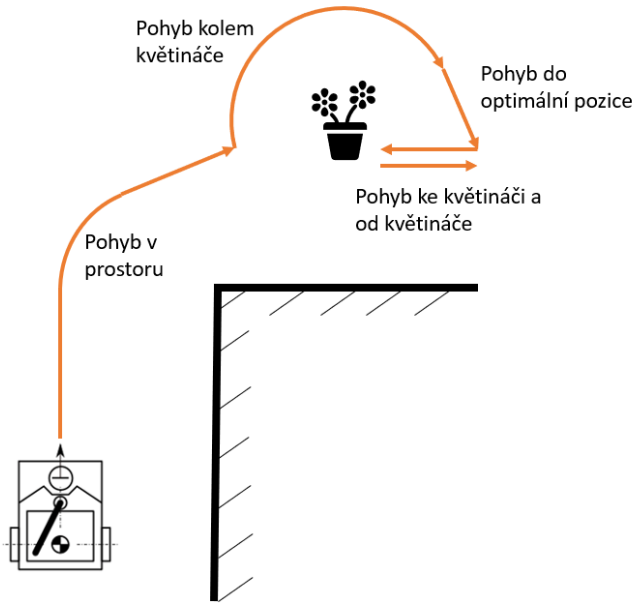
6 POHYBY ROBOTU

Následující kapitola se zabývá rozdělením pohybu robotu do jednotlivých částí, z důvodu různorodosti přístupů v určitých fázích pohybu a rozdělení pohybu do takových částí, které budou fungovat po celou dobu bez nutnosti jejich přerušení.

Pohyb robotu v prostoru, kde má zalévat rostliny, je komplikovaný velkým množstvím překážek, kterým se během své cesty musí robot vyhýbat, aby do nich nenarazil a dostal se do svého cíle. Na druhou stranu musí být schopen se přiblížit k rostlině na takovou vzdálenost, aby se nesrazil s květináčem a byl ji schopen zalít. Jelikož by se tento pohyb jako celek nedal jednoduše realizovat, byl rozdělen na více dílčích částí. Tyto části jsou následující:

a) Pohyb robotu v prostoru,
b) Pohyb identifikační,
c) Pohyb robotu do optimální pozice,
d) Pohyb robotu ke kytce,
e) Pohyb robotu od květiny.

V následující části bude popsán pohyb robotu, již rozdělen na jednotlivé části, aby bylo zhruba vidět, co která část vykonává (vyobrazeno na Obrázek 6-A).



Obrázek 6-A – Vizuální reprezentace jednotlivých částí pohybu

a) Pohyb robotu v prostoru
   Pohyb robotu spočívá v pohybu robotu v prostoru, kde přejíždí po místnosti od jedné rostliny ke druhé, aby je mohl zalít. Při tomto pohybu se vyhýbá překážkám, dokud nedojede do cílové pozice v blízkosti rostliny, která je v pořadí.

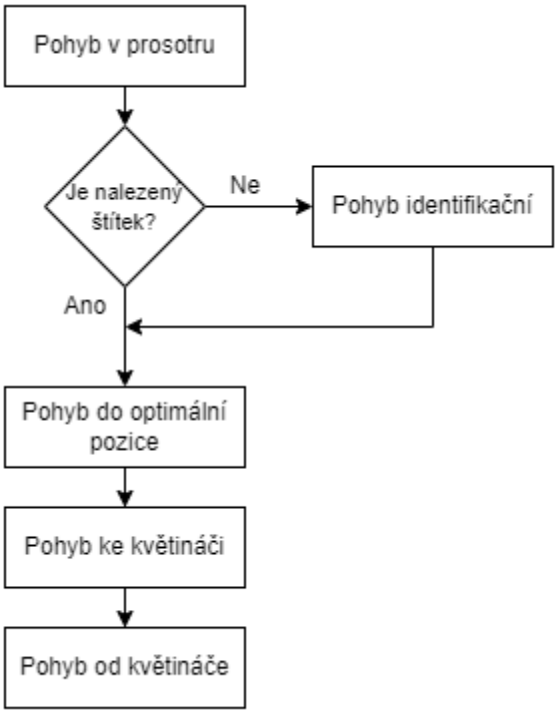
b) Pohyb identifikační
   V následující části má robot za úkol nalézt rostlinu a identifikovat ji podle štítku umístěného na květináči. Pokud by nebyl štítek vidět, pokusí se robot danou rostlinu objet a podívat se na květináč z jiného úhlu, aby dokázal spatřit, kde se štítek nachází.

c) Pohyb robotu do optimální pozice
Po úspěšném nalezení štítku a identifikovaní, zda se jedná o správnou rostlinu, odjede robot do optimální pozice, ze které uvidí dobře na rostlinu a bude mít kolem sebe prostor, aby mohl natočit rameno určené pro zalévání a do ničeho s ním nenarazil.

d) Pohyb robotu ke květináči
V tuto chvíli stojí robot otočený směrem k rostlině, před ním se nachází volný prostor a může se vydat směrem k ní. Při tomto pohybu se robot pohybuje pomalu a průběžně se natáčí, aby se dostal k rostlině. Při pohybu má natažené rameno před sebou, aby byl schopen rozpoznat, kdy se nachází zalévacím ramenem nad květináčem. Po nalezení optimální pozice dojde k zastavení robotu a vykoná se zalití rostliny.

e) Pohyb robotu od květináče
Poslední částí tohoto pohybu je pohyb robotu od rostliny. V tuto chvíli je již rostlina zalita a robot se potřebuje od ní oddálit. Potřebuje si vytvořit prostor pro manévrování v případě zadání nové rostliny, kterou má zalít, a pro navrácení zalévacího ramene do počáteční pozice.

Jednotlivé pohyby ve většině případů navazují přímo na sebe, výjimku tvoří pohyb identifikační, který může být vynechán. Dojde k tomu, pokud je robot schopen nalézt štítek na květináči. V tomto případě není důvod vykonávat identifikační pohyb a zbytečně se snažit objet rostlinu. Přehled všech druhů pohybů a jejich návaznost je znázorněna na následujícím schématu.



Obrázek 6-B – Schéma návaznosti pohybů

6.1 Pohyb robotu v prostoru

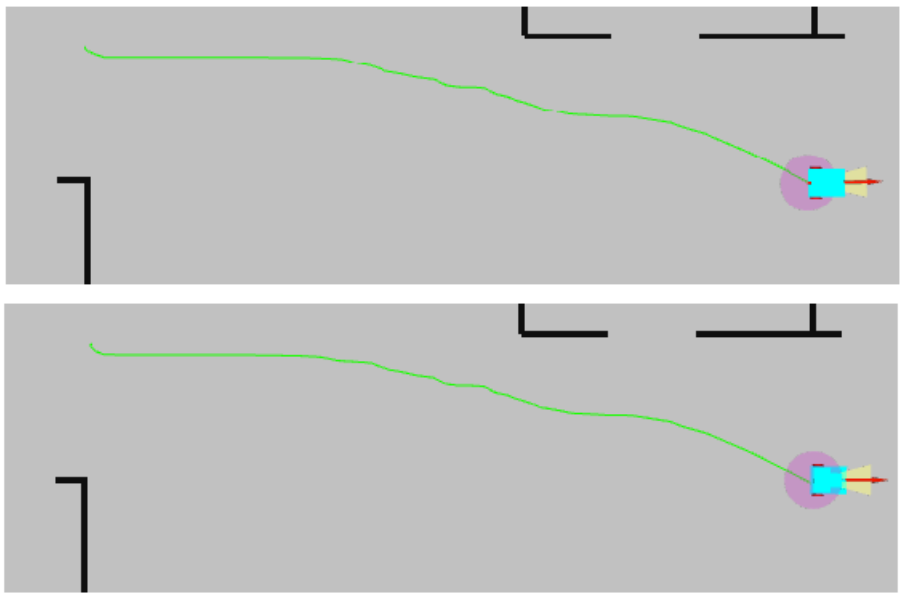
Základním druhem pohybu pro robotu je pohyb v prostoru. Tento pohyb slouží pro přesun robotu mezi rostlinami a zásobníkem vody. Skládá se ze dvou částí – globálního a lokálního plánovače. Globální plánovač se stará o nalezení cesty mezi dvěma body za využití známé statické mapy okolí. Lokální

plánovač se naopak stará o vyhýbání se dynamickým překážkám v cestě a překážkám, které nejsou zavedeny ve statické mapě okolí. Mezi jeho další úkoly patří určení rychlostí pro motory.

Pohyb robotu v prostoru funguje jako ostatní pohyby v této kapitole na principu akce. Kde při zavolání akce klientem se předá serveru identifikační číslo (ID) požadovaného objektu, ke kterému má dojet. Aby robot z identifikačního čísla poznal, o jaký objekt se jedná, přesněji o to, kde se nachází, potřebuje získat z databáze dané informace. Po převzetí polohy objektu se předá globálnímu plánovači a spustí se plánování pohybu. Během pohybu robotu prostorem poskytuje sever klientovi zpětnou vazbu v procentech progresu jízdy prostorem k cíli.

6.1.1 Globální pohyb robotu v prostoru

Pro prvotní testování byl použit balíček *move_base*, který se používá pro plánování pohybu robotu v prostoru. Plánování pohybu je v tomto balíčku vykonáváno pomocí globálního a lokálního plánovače. Jako globální plánovač má *move_base* k dispozici dva plánovače, jedná se o nejpoužívanější plánovač A* a Dijkstrův algoritmus. Podle teorie by měl být algoritmus A* rychlejší než Dijkstrův algoritmus, z důvodu, že A* neprohledává celý prostor. Při testování v simulaci se zjistilo, že při použití obou algoritmů byly výsledné naplánované trasy téměř totožné a časový rozdíl výpočtu plánování cesty byl při použití na malém prostoru vytvořeném v simulaci zanedbatelný. Z tohoto důvodu byl použit jako defaultní plánovač Dijkstrův algoritmus.



Obrázek 6-C – Srovnání algoritmů používaných pro globální plánovače v balíčku move_base, A* (nahoře) a Dijkstrův algoritmus (dole)

Jelikož výsledná naplánovaná trasa globálního plánovače byla bezchybná a podařilo se mu najít vhodnou cestu z počáteční do cílové pozice, byl tento algoritmus využit ve zbývající části práce.

Globální plánovač funguje pro plánování cesty na mřížce, reprezentace prostoru je vytvořena pomocí balíčku *costmapě_2d*, jenž vytváří mřížku obsazenosti (occupancy grid), kde je zobrazen pravděpodobnostní výskyt překážek pro každou buňku mřížky v prostoru. Od místa, kde je 100% výskyt překážky, pravděpodobnost klesá se vzrůstající vzdáleností. Costmapy jsou aktualizovány za pomoci

sensorů, kdy jsou jednotlivé překážky odebírány a přidávány do mapy. Jak globální, tak lokální plánovač má vlastní costmapu, která se využívá při plánování pohybu.

Balíček *move_base* obsahuje i sadu metod (recovery behaviors), které mu pomohou v případě, že se robot zasekne a plánovač není schopen vytvořit novou cestu. Robot se začne otáčet dokola, aby aktualizoval informace o svém okolí. Pokud se mu podaří naplánovat novou cestu, pokračuje robot v pohybu. V opačném případě využije robot dalších možných způsobů, jak se dostat z této situace. [23] [34] [24]

Lokální plánovač, který je v tomto balíčku k dispozici, je Dynamic Window Approach (DWA). Jeho porovnání s vlastní implementací lokálního plánovače je uvedeno v kapitole 6.1.2.

6.1.2 Lokální pohyb robotu v prostoru

Po naplánování trasy robotu z pozice, kde se nachází do cílové polohy, je potřeba přimět robota, aby tuto trasu následoval, a při tom se vyhýbal překážkám. Překážky statické, které jsou již zavedené v mapě, bere robot v úvahu při plánování globální trasy. Naopak překážkám dynamickým se musí robot vyhýbat při pohybu do cílové pozice. Pro pohyb robota je potřeba určit rychlosti motorů, aby se pohyboval správným směrem a přiměřenou rychlostí. O všechny tyto problémy se stará lokální plánovač.

Pro správné fungování lokálního plánovače by měly být splněny následující předpoklady:

- následování robotu předem vytyčené globální trasy,
- vyhýbání se překážkám,
- udržování předem zvolené rychlosti,
- přibližování se k cíli.

Algoritmy, které lze využít pro lokální plánování pohybu, jsou popsány v kapitole 3.3, společně s jejich výhodami i nevýhodami.

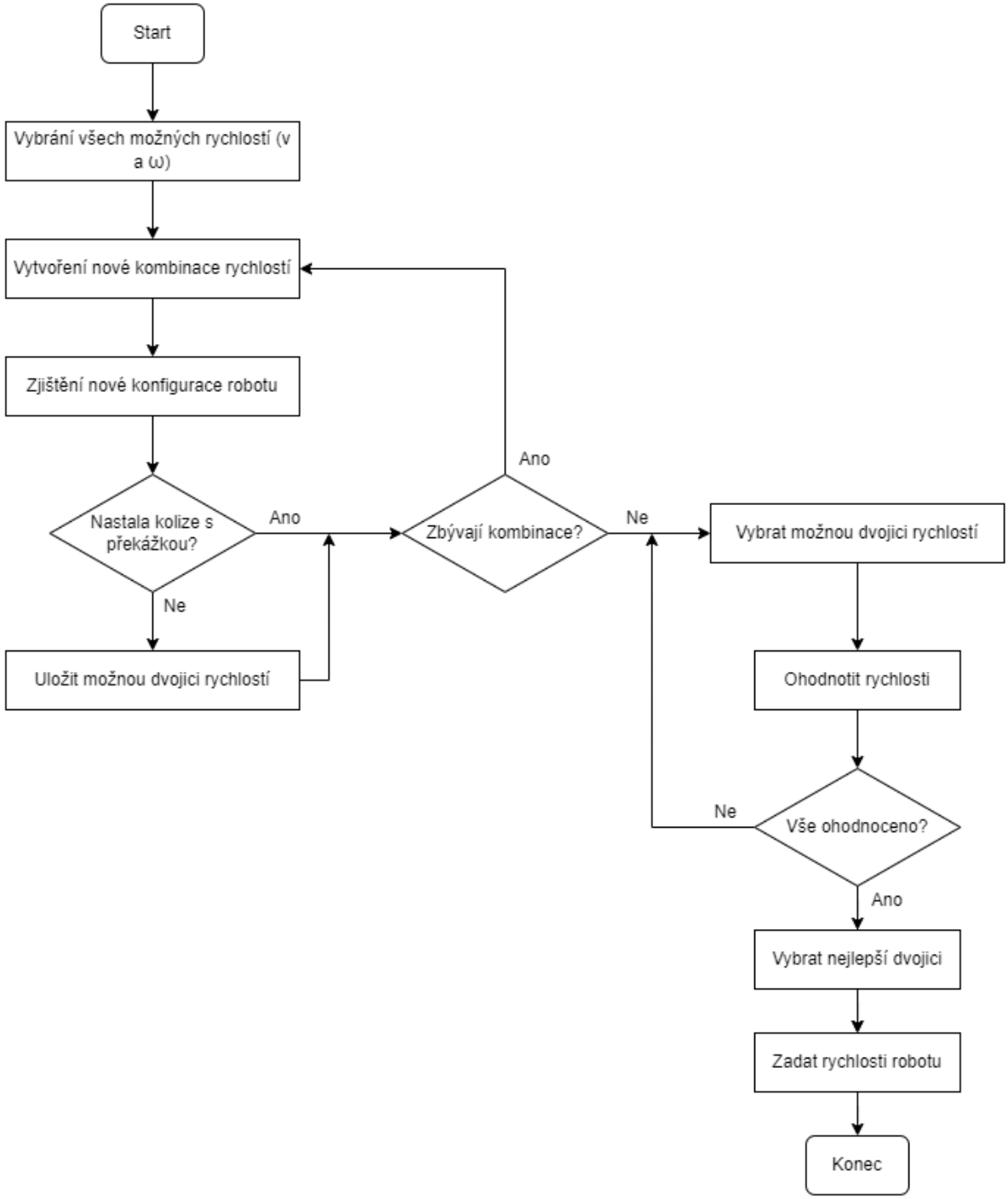
Jako nejlepší možnost se jevilo využití algoritmu Dynamic Window Aproach (DWA).

Vytvoření vlastního lokálního plánovače

Dynamic Window Aproach je lokální plánovač založený na rychlostech robotu. Během tohoto algoritmu se určuje optimální rychlost pro robota, tak aby byl schopen bez kolize s překážkami dojet do cílové polohy. Algoritmus převádí kartézské souřadnice x, y na rychlosti v, ω, které jsou zadávány mobilnímu robotu.

Hlavním principem algoritmu je vybrání všech možných rychlostí, kterých je robot schopen dosáhnout v určitém časovém intervalu vzhledem ke své dynamice a vybrat z nich optimální dvojici. Mezi skupinu všech možných rychlostí se řadí rychlosti, které vytváří bezpečnou cestu, což znamená, že je robot schopen zastavit bezpečně před překážkou, pokud by směrem k ní směřovala cesta robotu. Optimální rychlost je zvolena tak, aby se maximalizovala vzdálenost robotu od překážky, robot se pohyboval optimální rychlostí a dostal se co nejblíže ke svému cíli.

V následující části bude popsán program lokálního plánovače, princip funkčnosti je znázorněn na vývojovém diagramu.



Obrázek 6-D – Schéma popisující jeden cyklus lokálního plánovače DWA

Nejdříve je potřeba vybrat všechny možné rychlosti, kterých může robot dosáhnout v následujícím časovém intervalu. Při vybírání rychlostí se bere v potaz maximální rychlost robotu a jeho maximální zrychlení. Při výběru minimální rychlosti robotu nelze vybrat zápornou rychlost z toho důvodu, protože sensory robotu nejsou schopny vidět, co se děje za ním a byla by vysoká šance, že narazí do překážky, o které neměl tušení, že se za ním nachází. Zákaz couvání je vyvážen možností robotu se otočit na místě.

Výběr všech možných rychlostí:

```
linear_upper_limit = min(LIN_MAX, lin_last + LIN_ACC_MAX * DELTA_TIME)
linear_lower_limit = max(0, lin_last - LIN_ACC_MAX * DELTA_TIME)
angular_upper_limit = min(ANG_MAX, ang_last + ANG_ACC_MAX * DELTA_TIME)
angular_lower_limit = max(-ANG_MAX, ang_last - ANG_ACC_MAX * DELTA_TIME)
```

, přičemž LIN_MAX a ANG_MAX jsou maximální lineární a úhlové rychlosti a LIN_ACC_MAX a ANG_ACC_MAX jsou maximální zrychlení, DELTA_TIME je pak určitý časový úsek.

V následující části se určí pro všechny možné kombinace lineárních a úhlových rychlostí poloha, kam se robot dostane za určitý časový úsek. Tato poloha je vypočítaná pomocí následujících rovnic:

$$x_{new} = x + v \cdot \Delta t \cdot \cos(\theta + \omega \cdot \Delta t) \quad (16)$$

$$y_{new} = y + v \cdot \Delta t \cdot \sin(\theta + \omega \cdot \Delta t) \quad (17)$$

$$\theta_{new} = \theta + \omega \cdot \Delta t \quad (18)$$

, kde v je lineární rychlost, ω je úhlová rychlost, Δt je časový úsek, x, y, θ jsou aktuální souřadnice a natočení robotu a $x_{new}, y_{new}, \theta_{new}$ je nová pozice a natočení, které bude mít robot, pokud by se pohyboval rychlostmi v a ω po dobu Δt.

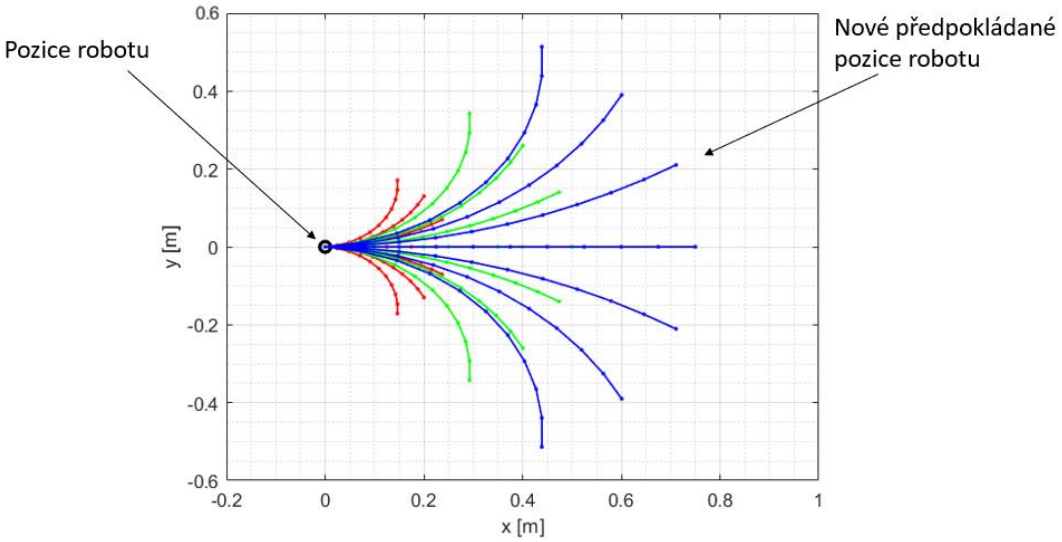
Ukázka části programu:

```
for i in range(config.TIME_STEPS):

      delta_t = config.DELTA_TIME/config.TIME_STEPS

      delta_theta = angular*delta_t
      delta_x = linear*delta_t*math.cos(new_pos[2] + delta_theta)
      delta_y = linear*delta_t*math.sin(new_pos[2] + delta_theta)

      new_pos = [new_pos[0] + delta_x,
                 new_pos[1] + delta_y,
                 new_pos[2] + delta_theta]
```



Obrázek 6-E – Znázornění všech předpokládaných pozic. kam se robot může dostat

Pokud mobilní robot s touto kombinací rychlostí narazí do překážky nebo se dostane do takové pozice, kdy by nebyl schopen zabrzdit před překážkou, je tato kombinace rychlostí vyřazena z možných rychlostí.

V posledním kroku je zapotřebí vybrat ze všech možných rychlostí tu nejvýhodnější. Všechny možnosti jsou ohodnoceny a je vybrána dvojice rychlostí, která je předána robotu. Rychlosti jsou ohodnoceny podle těchto kritérií:

- Natočení k cíli (*scoreHeadingGoal*),
- Natočení k následující pozici (*scoreHeadingTarget*),
- Vzdálenost od nejbližší překážky (*scoreClearence*),
- Optimální rychlost (*scoreDesiredLinearVelocity*),
- Vzdálenost od následující pozice (*scoreDistanceTarget*).

Následně budou rozepsána jednotlivá kritéria. U většiny případů platí, že čím nižší hodnocení, tím lepší.

- Natočení k cíli (*scoreHeadingGoal*)

Toto kritérium zabezpečuje, aby u cíle byl robot natočen správným směrem. Hodnotí se tedy odchýlení natočení robota od cílové pozice.

- Natočení k následující pozici (*scoreHeadingTarget*)

Pro následování globální trasy, a tedy i jízdu ve správném směru, je ohodnoceno natočení robotu vzhledem k následujícím pozicím globální trasy. Jedná se o snahu, aby robot jel ve směru, kam vede globální trasa. Robot bude preferovat rychlosti, které budou směřovat správným směrem.

- Vzdálenost od nejbližší překážky (*scoreClearence*)

Pro polohu, kam by se robot dostal při daných rychlostech, se nalezne nejbližší překážka. Pokud by se tato překážka nacházela v blízkosti robota, bude ohodnocena. V opačném případě bude hodnocení nejnižší, hodnocení je v tomto případě otočeno a na konci se odečítá od celkového.

```
if distNearestObstacle < minObstacleDistance:
     clearance = minObstacleDistance - distNearestObstacle
else:
     clearance = 0
```

- Optimální rychlost (*scoreDesiredLinearVelocity*)

Ohodnocení rychlosti, kterou by se robot pohyboval, probíhá následovně. Jako optimální rychlost je většinou zvolena maximální rychlost robotu. Při přiblížení robota k cíli je optimální rychlost snížena. Pro ohodnocení rychlosti je použita následující rovnice:

$$scoreDesiredLinearVelocity = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\left(\frac{(x-\mu)^2}{2\cdot\sigma^2}\right)} \tag{19}$$

, kde μ reprezentovalo požadovanou rychlost, x aktuální rychlost a σ směrodatnou odchylku (neboli mění šířku grafu).

- Vzdálenost od následující pozice (*scoreDistanceTarget*)

Aby robot následoval předem vytvořenou globální trasu, je ohodnocena následující pozice robotu a její vzdálenost od globální trasy.

Pro ulehčení celkového hodnocení jsou všechny předchozí kritéria normalizována na hodnoty (0-1), kde nejhorší výsledek je nahrazen hodnocením 0 a nejlepší hodnocením 1. Všechny hodnoty mezi těmito limity jsou přepočítány, jak je zobrazeno v následující části programu.

```
if maximum - minimum == 0:
      score_norm = [0.0 for i in range(len(score))]
else:
      score_norm = (score - minimum) / (maximum - minimum)
```

Finální ohodnocení je vážený součet (popřípadě rozdíl) všech normalizovaných kritérií.

$$\begin{aligned} cost = \; & + \alpha \cdot scoreHeadingTarget \\ & - \beta \cdot scoreClearance \\ & + \gamma \cdot scoreDesiredLinearVelocity \\ & + \delta \cdot scoreHeadingGoal \\ & + \epsilon \cdot scoreDistanceTarget \end{aligned} \tag{20}$$

, kde parametry $\alpha, \beta, \gamma, \delta, \epsilon$ jsou váhy jednotlivých ohodnocení.

Ohodnocení, jak blízko se robot nachází od překážky, se bere jako záporná hodnota, z důvodu, že se jedná o nechtěnou situaci.

Před prvním spuštěním cyklu lokálního plánovače je potřeba počkat, dokud nejsou k dispozici všechna potřebná data pro jeho funkčnost. Při dalších cyklech se kontroluje, zda byla některá data aktualizována a jsou použita v následujícím cyklu.

Po porovnání existující varianty lokálního plánovače z balíčku *move_base* a vlastní interpretace vybíral vlastní vytvořený algoritmus lepší rychlosti než lokální plánovač od *move_base*.

6.2 Pohyb identifikační

Pohyb identifikační slouží k objetí květináče ze všech stran a nalezení štítku k identifikaci správné rostliny. Tento pohyb se využije, pokud není štítek viditelný po příjezdu robotu ke květináči. Pokud by byl štítek identifikován, je tento pohyb vynechán.

Požadavky:

1. Místo, které je potřeba objet nemusí obsahovat překážku.
2. Robot musí při jízdě vidět na květináč:
    a. objíždění květináče s určitou minimální vzdáleností pro viditelnost celého květináče,
    b. nemůže se nacházet překážka mezi robotem a květináčem pro lepší viditelnost,
    c. kamera se nedokáže příliš natáčet (maximálně 30°), robot se musí při cestě natáčet ke květináči.

První požadavek je zaveden z důvodu, že květináč může být zavěšený ve vzduchu nebo podepřen tak, že je špatně detekován sensory. Druhá podmínka zaručuje, že robot vidí na štítek umístěný na květináči. Robot nemůže objíždět rostlinu příliš blízko květináči, kamera musí mít dostatečný úhel pohledu na květináč. Kvůli umístění kamery pod lidarem se nemůže kamera natáčet o 90° na obě strany. Pro nezastíněný pohled na květináč, se musí robot pohybovat kolem květináče a v určitých intervalech se otáčet ke květináči, aby se mohl podívat po štítku. Z tohoto vychází poslední část, kdy se v tomto místě mezi robotem a květináčem nesmí nacházet překážka.

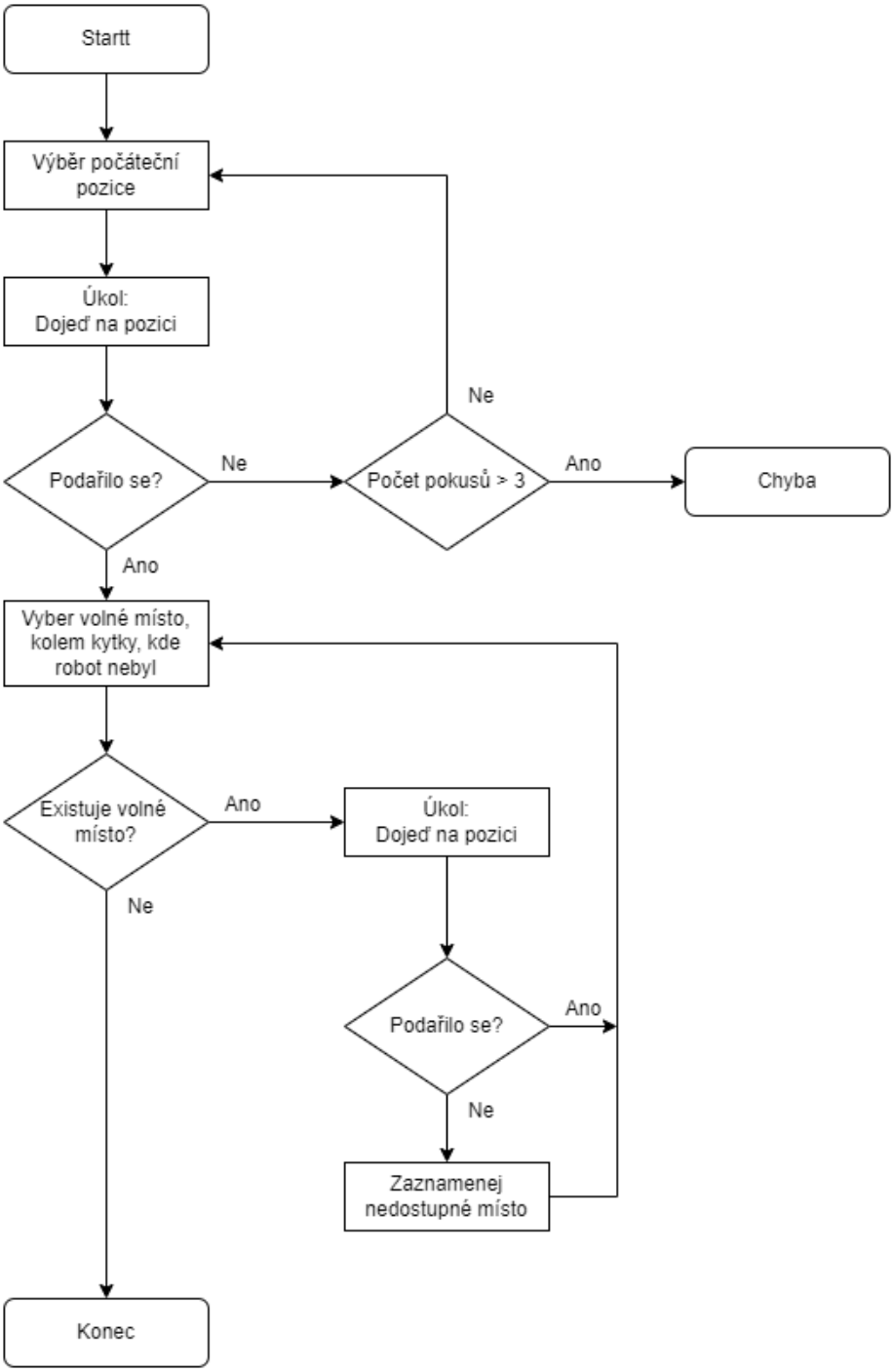
Obrázek 6-F – Štítek obsahující ID rostliny

Návrhy:

Využití upraveného algoritmu Tangent Bug Algorithm by v tomto případě nebylo možné, protože při objíždění květináče by se robot nacházel v jeho těsné blízkosti a zároveň by při jízdě na nej neviděl.

Dalším návrhem bylo vytvoření kružnice o určitém poloměru kolem objížděného místa a následného vybrání výseče, která se nachází před robotem a rozprostírá se od jedné překážky ke druhé. Z tohoto úseku by bylo vybráno N bodů posunutých o určitý úhel a na tyto místa by byl robot poslán.

Poslední návrh je podobný předchozímu. Robot by se přesunul blíže k objížděnému objektu. Pomocí lidaru a dat z mapy si vytvořil okolí rostliny s minimálním poloměrem (zaručení viditelnosti květináče) a maximálním poloměrem (omezení vzdálení robotu od květináče). V tomto úseku se vyberou místa, kde se nachází překážky, tedy kde robot nevidí na květináč. K hranicím těchto objektů se přidá menší odchylka pro oddálení robotu od překážek. Pro úhel, pod kterým je květináč vidět, se vybere místo, které se nachází ve volném prostoru a pošle se tam robot, aby z tohoto místa prozkoumal květináč. Okolí tohoto úhlu bude následně přidáno do prozkoumané oblasti (oblasti s překážkou) a vybere se nová pozice k prozkoumání. Po prozkoumání všech možných pozic se ukončí identifikační pohyb.

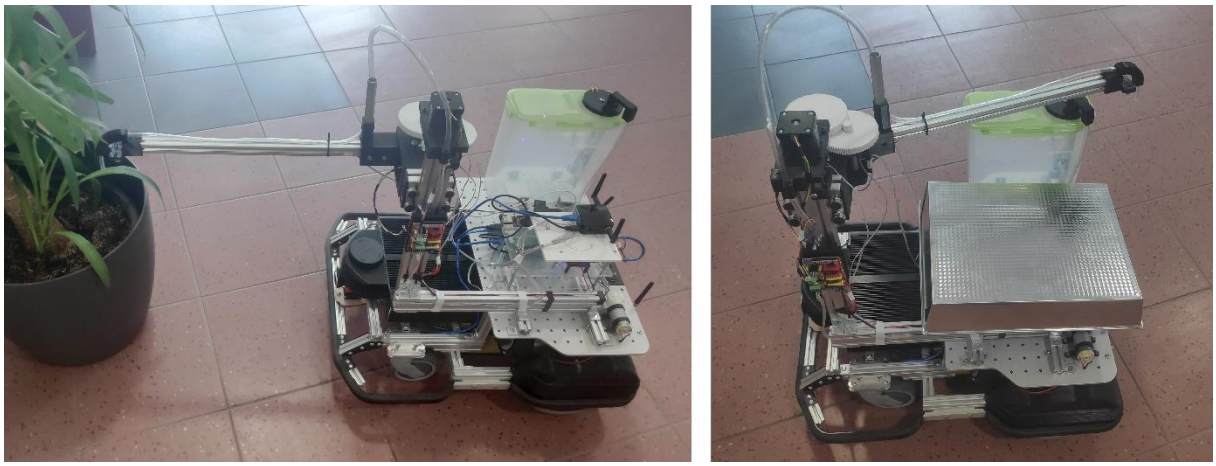


Obrázek 6-G – Schéma algoritmu pro identifikační pohyb

6.3 Pohyb robotu do optimální pozice

Pohyb robotu do optimální pozice slouží k přesunu robotu do pozice, ze které se robot vydá na cestu směrem ke květináči. Výběr optimální pozice je popsán v diplomové práci „Komplexní návrh zalévacího mobilního robotu“ [1].

Optimální pozice je zvolena takovým způsobem, aby měl robot kolem sebe prostor a mohl nastavit zalévací rameno před sebe a připravil se na zalití rostliny.



Obrázek 6-H – Znázornění pozic zalévacího ramene, robot s nataženým ramenem v zalévací pozici (vlevo) a robot s uschovaným ramenem v defaultní pozici (vpravo)

Poté co byla zvolena optimální pozice, je zadána robotu, aby do požadované pozice dojel. Jelikož se pozice nenachází v těsné blízkosti okolních překážek, byl na plánování a vykonání cesty do této pozice využit plánovač, který se používá pro plánování pohybu v prostoru (Kapitola 6.1). Tento plánovač je lehce poupravený a pro jeho spuštění je potřeba pouze zadat cílovou pozici.

6.4 Pohyb robotu ke květináči

Pohyb ke květináči se stará o to, aby se robot dostal ke květináči s vysunutým ramenem do takové vzdálenosti, kdy se bude zalévací rameno umístěno nad květináčem a robot bude schopen úspěšně zalít rostlinu.

Na začátku pohybu se robot nachází v pozici, kdy je natočen směrem ke květináči, který má zalít, přičemž se mezi robotem a květináčem nenachází žádná překážka, takže je robot schopen k němu přímo dojet.

Před tím, než se robot rozjede, nastaví zalévací rameno před sebe do zalévací pozice. Na konci ramene se nachází ultrazvukový sensor, který kontroluje, kdy se konec ramene nachází nad květináčem a kdy má robot zastavit, aby mohl zalít rostlinu. Z toho vyplývá, že se robot zastaví, pokud se dostane zalévacím ramenem nad květináč. Další možností, kdy robot zastaví je tehdy, když se robot přiblíží na minimální vzdálenost ke květináči.

Přibližování robotu ke květináči probíhá na základě dat, které jsou získávána ze systému. Jedná se o vzdálenost robotu od květináče, přesněji vzdálenost robotu od štítku */marker_distance* a úhel natočení mezi robotem a štítkem */marker_angle*.

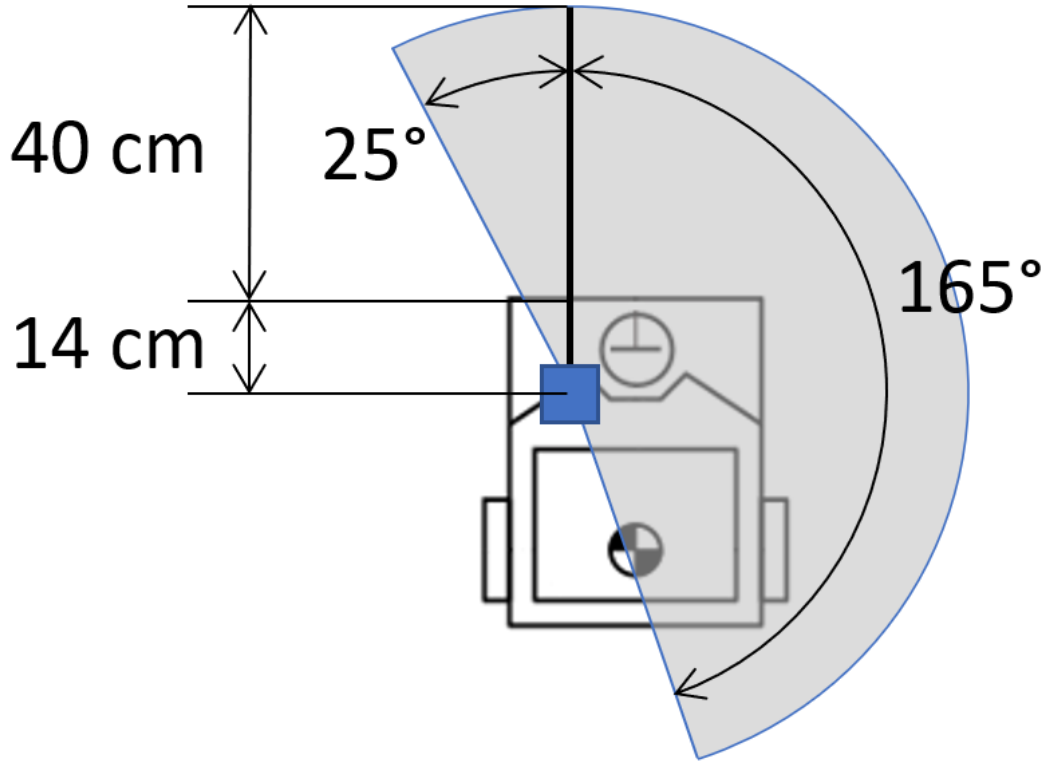
Jelikož se nenachází mezi robotem a květináčem překážka, jedná se o přímočarý pohyb robotu k rostlině. Z možnosti odklonění robotu od přímé cesty ke květináči byl vytvořen jednoduchý dvouhodnotový regulátor, který při odchýlení robotu od směru ke květináči začne robota otáčet, aby směřoval znovu správným směrem.

6.5 Pohyb robotu od květináče

Poté, co robot úspěšně zalil rostlinu, je zapotřebí vrátit rameno zpátky do původní pozice. Pro vytvoření volného prostoru kolem robotu, aby ramenem do ničeho nenarazil a aby se mohl lehce rozjet k dalšímu cíli, je potřeba robota dostat dál od květináče.

Jelikož robot nevidí za sebe, za pomoci sensorů a ani pomocí kamery, je nejvýhodnější pro robota odjet stejnou cestou, kterou ke květináči přijel. Pro pohyb od květináče by se tedy dal použít stejný princip, který je použitý pro navigaci v prostoru, ale z důvodu toho, že se robot nachází v těsné blízkosti květináče, hrozí jeho zaseknutí. Další možností je otočení robota na místě, ale pro tento manévr potřebuje robot kolem sebe větší prostor, jinak by mohl narazit do překážky nataženým zalévacím ramenem. Poslední možností je, aby robot začal couvat a poodjel do bezpečné oblasti, tedy jel stejnou cestou, kterou se dostal ke květináči.

Na následujícím obrázku (Obrázek 6-I) je znázorněna oblast, u které je zapotřebí, aby byla průchozí pro přesun ramene do defaultní polohy (polohy, kdy je rameno uschováno nad robotem)



Obrázek 6-I – Schéma znázorňující dosah zalévacího ramene

Jako nejlepší řešení byla vybrána poslední možnost. Jediným předpokladem pro tento pohyb je to, že se za robotem neobjevila žádná nová překážka, zatímco se nacházel u rostliny.

Po spuštění pohybu robotu od rostliny začne robot pomalu couvat a po ujetí předem zadané vzdálenosti se robot zastaví a ukončí pohyb.

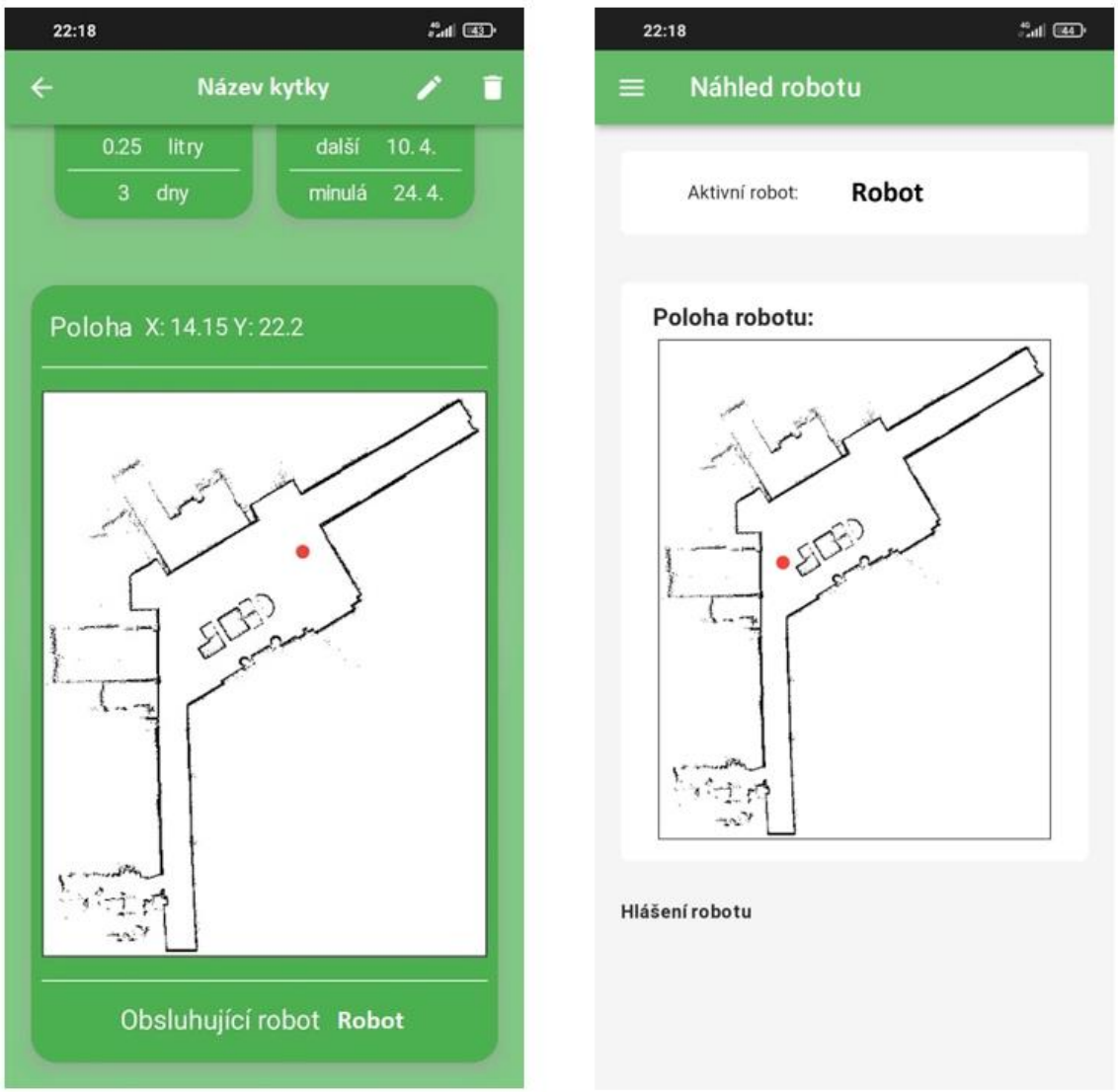
7 KOMUNIKACE ROBOTU S APLIKACÍ

V následující kapitole bude popsána komunikace, která probíhala mezi robotem a aplikací pro předávání informací ohledně mapy prostředí a polohy robotu.

7.1 Zobrazení polohy robotu

Pro lepší vizualizaci, kde se rostlina nebo robot v prostoru nachází, byla do aplikace přidána mapa okolí, po kterém se robot pohybuje. Mapa byla převzata z ROSu, kde se využívá pro navigaci a pohyb robota. Pro odesílání mapy do aplikace byl vytvořen servis, který pokud se zavolá, přečte topic obsahující mapu, zaznamená důležité informace jako jsou rozměry mapy (šířka a výška) a okrajové pozice (pozice x a y). Spolu s daty obsahující mapu je odešle do aplikace. Pro snížení velikosti odesílané zprávy jsou data popisující mapu zašifrována.

Výsledná mapa prostoru je zobrazena na následujícím obrázku (Obrázek 7-A)



Obrázek 7-A – Vizualizace mapy okolí robotu v aplikaci a příklad znázornění polohy rostliny (vlevo) a pozice robotu (vpravo)

7.2 Zobrazení mapy prostředí

Pro správné zobrazení robotu v mapě byl vytvořen skript, který se stará o získávání pozice robotu v prostoru a odesílání pozice formou topicu do systému. Pozice robotu je pak přístupná pro jakékoliv použití. Pozice robotu byla odesílána ve formátu (x,y,θ), kde x a y značí souřadnici polohy robotu vzhledem k levému spodnímu okraji mapy a θ jeho natočení.

Mapa, která se zobrazí v aplikaci, je znázorněna na obrázku v předchozí podkapitole (Obrázek 7-A)

8 OTESTOVÁNÍ NA REÁLNÉM ROBOTU

V následující kapitole bude popsáno testování plánovaní pohybu na reálném robotu a zhodnocení funkčnosti navržených algoritmů na robotu v reálných podmínkách.

Otestování jednotlivých pohybů bylo provedeno na reálném robotu, nejdříve na zmenšené verzi bez zalévací platformy, na robotu Leela a později na robotu Breach se zalévací platformou. Robot Leela byl použit z důvodu nedostupnosti robotu Breach v první fázi testování.

První algoritmus, který byl testován na robotu Leela, byl algoritmus pro plánování pohybu robotu ke květináči a plánování pohybu robotu od květináče. Robot se postavil do určité vzdálenosti od květináče a spustilo se plánování pohybu ke květináči. Robot se pomalu rozjel konstantní rychlostí směrem k rostlině a v případě, kdy se vychýlil od přímé dráhy, se zastavil a zkorigoval své natočení směrem k rostlině. Po příjezdu do určité vzdálenosti od štítku se robot zastavil. Následně se provedlo testování pohybu robotu od rostliny. Robot v tomto případě pouze začal couvat a po ujetí definované vzdálenosti se zastavil.

Obrázek 8-A – Testování na robotu Zaleela

Jakmile byl robot Breach, připraven k použití, přešlo se k testování na tohoto robota, robot Breach byl navíc vybaven zalévací platformou. Testování pohybu od/ke květináči probíhalo stejně jako u předchozího robotu i se stejnými výsledky. Pro pohyb robotu v prostoru byl použit defaultní plánovač z důvodu, že v této fázi testování nebyla ještě odhalena chyba v simulaci a pohyb identifikační nebo-li pohyb kolem rostliny nebyl v tuto chvíli ještě implementován.

Obrázek 8-B – Testování pohybu na robotu Breach

Pohyb robotu započal zadáním ID rostliny, kterou má zalít. Pomocí plánování pohybu v prostoru dojel robot do blízkosti rostliny a při přiblížení a detekování štítku na květináči byl pohyb v prostoru přerušen. Robot následně dojel do optimální pozice, nastavil si zalévací rameno a vydal se směrem ke květináči zalít rostlinu. Pozice rostliny byla získávána za pomoci lidaru a zpracováním obrazu z kamery. Při detekování hrany květináče ultrazvukovým senzorem byl vyslán příkaz robotu k zastavení. Po zastavení zalil robot rostlinu, poté poodjel od rostliny pryč a přesunul rameno do základní pozice.

8.1 Zhodnocení

V následující podkapitole bude shrnuto hodnocení jednotlivých částí pohybu reálného robotu.

S pohybem v prostoru neměl robot problém a pohyboval se volným prostorem bez narážení do překážek. Jediný problém nastal v zúženém prostoru, kde robot v malém počtu případů narazil do zdi.

Pohyb robotu do optimální pozice ne vždy probíhal vhodným způsobem. Například místo malé korekce natočení se robot otočil celý dokola, ale pokaždé byl schopen se do optimální pozice dostat.

Pohyb robotu k rostlině probíhal bez problémů, stejně tak pohyb robotu od květináče.

9 ZÁVĚR

Nejprve by bylo vhodné shrnout hlavní cíle této práce. Prvním cílem bylo navrhnout architekturu plánování pohybu robotu během zalévání pokojových rostlin. Pohyb robotu jako celek byl rozdělen do několika dílčích částí. Jedná se o pohyb v prostoru, pohyb identifikační, pohyb robotu do optimální pozice, pohyb robotu ke květináči a pohyb robotu od květináče. Dalšími cíli bylo navržení metody globálního plánování pohybu robotu v prostoru a navržení metody lokálního plánování pohybu v oblasti zalévání. Předposledním cílem bylo navržení metody tzv. mikroplánování kolem vybrané rostliny pro navigaci do požadované zalévací pozice. Neboli umožnění robotu dostat se do takové blízkosti ke květináči, aby byl schopen zalít rostlinu nebo pokud to potřebuje rostlinu objet. Navržené algoritmy je následně potřebné integrovat do řídícího systému. Poslední částí je zhodnocení a ověření funkcionalit vybraného řešení.

První část této práce se zabývá rešerší různých druhů algoritmů použitelných pro plánování pohybu robotu a algoritmů pro vyhýbání se překážkám. Na základě rešerše byly vybrány vhodné algoritmy pro globální i lokální plánování pohybu. Pro globální plánovač byl vybrán Dijkstrův algoritmus a pro lokální plánovač Dynamic Window Aproach (DWA). Druhá část rešerše byla věnována seznámení se s Frameworkem ROS.

V následující části byl představen robot Breach, který se stal základem zalévacího robotu. Předtím než byl robot k dispozici, probíhalo testování programů v simulačním prostředí. Pro simulaci byly použity dva programy. Nejdříve byl použit program Stage pro vytvoření první, jednoduché simulace, na které se dalo odzkoušet a vyladit vytvořený algoritmus plánovače. Ve chvíli, kdy byl připraven pro použití program Gazebo, přešla simulace do tohoto programu.

Pohyb robotu bylo potřeba rozdělit do několika dílčích částí. Pohyb v prostoru slouží k přesunu robotu mezi jednotlivými rostlinami, které je potřeba zalít, a zdroji vody. Skládá se z globálního a lokálního plánovače. Jako globální plánovač byl využit defaultní plánovač z balíčku *move_base,* jelikož jeho naplánovaná trasa, byla ve většině případů bezchybná. Naproti tomu jako lokální plánovač byla použita vlastní implementace algoritmu Dynamic Window Approach (DWA). Pohyb identifikační byl vytvořen pouze jako koncept, jeho účelem je zajistit nalezení identifikačního štítku umístěného na květináči. Pohyb do optimální pozice slouží k přesunu robotu před rostlinu, do definované vzdálenosti a úhlu natočení. Úkolem pohybu robotu k rostlině je přesunout robota do zalévací pozice a úkolem pohybu robota od květináče je naopak oddálení robotu od rostliny, pro bezpečné otočení zalévacího ramene do základní pozice.

V další části je představen důvod komunikace robotu s aplikací, a to za účelem předávání informací o mapě prostředí a poloze robotu pro jejich vizualizaci v aplikaci.

Závěrečnou část této práce tvoří testování algoritmů implementovaných do reálného systému. První fáze testování probíhala na robotu Leela, a později na robotu Breach. Při pohybování robotu v prostoru se robot úspěšně vyhýbal překážkám, po nalezení identifikačního štítku se mírnými problémy dostal do optimální polohy, odtud dojel k rostlině a úspěšně ji zalil a následně poodjel od rostliny pryč. Při pohybu v prostoru měl robot problém s úzkými prostory, při pohybu do optimální pozice se robot v několika případech nejprve celý otočil.

Na závěr lze konstatovat, že se povedlo vytvořit robota, který je schopen úspěšně se pohybovat v prostoru, přijet k rostlině a úspěšně ji zalít.

10 SEZNAM POUŽITÝCH ZDROJŮ

[1] PODOLINSKÝ, Ondřej. *Komplexní návrh zalévacího mobilního robotu* [online]. Brno, 2022 [cit. 2022-05-20]. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/140183. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Jiří Krejsa.

[2] SLADKÝ, Jiří. *Detekce a klasifikace objektů zájmu zalévacího robotu zpracováním obrazu* [online]. Brno, 2022 [cit. 2022-05-20]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/140181. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Jiří Krejsa.

[3] BAJER, Jan. *Návrh a realizace komunikačního rozhraní autonomního mobilního robotu* [online]. Brno, 2022 [cit. 2022-05-20]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/140227. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Stanislav Věchet.

[4] VIZVÁRY, Peter. *Návrh zalévacího modulu pro mobilní robot* [online]. Brno, 2022 [cit. 2022-05-20]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/140184. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Jiří Krejsa.

[5] SEDLÁK, V. *Plánování cesty mobilního robotu pomocí mravenčích algoritmů* [online]. Brno, 2011 [cit. 2022-05-20]. Dostupné z: https://dspace.vutbr.cz/xmlui/handle/11012/17176. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství. Vedoucí práce RNDr. JiříDvořák, CSc.

[6] BARTOZEL, Zdeněk. *Plánování cesty v reálném čase,* [online]. Brno, 2019 [cit. 2022-05-20]. Dostupné z: https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=193801. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí práce RNDr. Jiří Dvořák CSc.

[7] NORTHWESTERN ROBOTICS. Modern Robotics, Chapter 10.1: Overview of Motion Planning. In: *YouTube* [online]. [cit. 2022-05-20]. Dostupné z: https://www.youtube.com/watch?v=aC4LQuB4Cic

[8] NORTHWESTERN ROBOTICS. Modern Robotics, Chapter 10.2.3: Graphs and Trees. In: *YouTube* [online]. [cit. 2022-05-20]. Dostupné z: https://www.youtube.com/watch?v=QpOuyqd6nnc

[9] BURGARD, Wolfram, Cyrill STACHNISS, Maren BENNEWITZ a Kai ARRAS. *Introduction to Mobile Robotics: Robot Motion Planning* [online]. Freiburg im Breisgau: Albert-Ludwigs-Universität Freiburg [cit. 2022-05-20]. Dostupné z: http://ais.informatik.uni-freiburg.de/teaching/ss11/robotics/slides/18-robot-motion-planning.pdf

[10] NORTHWESTERN ROBOTICS. Modern Robotics, Chapter 10.5: Sampling Methods for Motion Planning (Part 1 of 2). In: *YouTube* [online]. [cit. 2022-05-20]. Dostupné z: https://www.youtube.com/watch?v=rKe6HO8LDu0

[11] NORTHWESTERN ROBOTICS. Modern Robotics, Chapter 10.5: Sampling Methods for Motion Planning (Part 2 of 2). In: *YouTube* [online]. [cit. 2022-05-20]. Dostupné z: https://www.youtube.com/watch?v=Ao7p_xiUu4s

[12] KARAMAN, Sertac a Emilio FRAZZOLI. Incremental Sampling-based Algorithms for Optimal Motion Planning. *Robotics Science and Systems VI* [online]. [cit. 2022-05-20]. Dostupné z: http://www.roboticsproceedings.org/rss06/p34.pdf

[13] NORTHWESTERN ROBOTICS. Modern Robotics, Chapter 10.6: Virtual Potential Fields. In: *YouTube* [online]. [cit. 2022-05-20]. Dostupné z: https://www.youtube.com/watch?v=8Vva0bnMIEI

[14] BOEING, Adrian. Dijkstra. In: *Adrian Boeing: Blog* [online]. 2010 [cit. 2022-05-20]. Dostupné z: http://adrianboeing.blogspot.com/2010/08/dijkstra.html

[15] BOEING, Adrian. A star (A*) pathplanning. In: *Adrian Boeing: Blog* [online]. 2010 [cit. 2022-05-20]. Dostupné z: http://adrianboeing.blogspot.com/2010/08/star-pathplanning.html

[16] FOX, D., W. BURGARD a S. THRUN. *The dynamic window approach to collision avoidance* [online]. **4**(1), 23-33 [cit. 2022-05-20]. ISSN 10709932. Dostupné z: doi:10.1109/100.580977

[17] BOEING, Adrian. Dynamic Window Algorithm motion planning. In: *Adrian Boeing: Blog* [online]. 2012 [cit. 2022-05-20]. Dostupné z: http://adrianboeing.blogspot.com/2012/05/dynamic-window-algorithm-motion.html

[18] *Dynamic Window Approach Tutorial* [online]. In: . 2020 [cit. 2022-05-20]. Dostupné z: https://www.youtube.com/watch?v=tNtUgMBCh2g

[19] ZOHAIB, M., M. PASHA, R. A. RIAZ, N. JAVAID, M. ILAHI a R. D. KHAN. *Control Strategies for Mobile Robot With Obstacle Avoidance*. 2013, **3**, 1027-1036.

[20] Vector Field Histogram. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2022 [cit. 2022-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Vector_Field_Histogram

[21] CHOSET, Howie. *Robotic Motion Planning: Bug Algorithms* [online]. [cit. 2022-05-20]. Dostupné z: http://www1.cs.columbia.edu/~allen/F15/NOTES/Chap2-Bug.pdf

[22] ÇULHA, Utku. *HW1 - Implementing Tangent Bug* [online]. [cit. 2022-05-20]. Dostupné z: http://www.cs.bilkent.edu.tr/~culha/cs548/hw1/

[23] ČEPL, Miroslav. *Návrh a implementace autonomního dokování mobilního robotu*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Jiří Krejsa.

[24] VÁVRA, P. *ROS framework utilization for autonomous mobile robot control system* [online]. Brno, 2019 [cit. 2022-05-20]. Dostupné z: https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=193943. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství. Vedoucí práce Ing. Martin Appel.

[25] ROS.org. *ROS Wiki: ROS/Introduction* [online]. Open Source Robotics Foundation [cit. 2022-05-20]. Dostupné z: http://wiki.ros.org/ROS/Introduction

[26] Robot Operating System. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2022 [cit. 2022-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Robot_Operating_System

[27] ROS.org. *ROS Wiki: distributions* [online]. Open Source Robotics Foundation [cit. 2022-05-20]. Dostupné z: http://wiki.ros.org/Distributions

[28] ROS.org. *ROS Wiki: Nodes* [online]. Open Source Robotics Foundation [cit. 2022-05-20]. Dostupné z: http://wiki.ros.org/Nodes

[29] ROS.org. *ROS Wiki: Topics* [online]. Open Source Robotics Foundation [cit. 2022-05-20]. Dostupné z: http://wiki.ros.org/Topics

[30] ROS.org. *ROS Wiki: Parameter Server* [online]. Open Source Robotics Foundation [cit. 2022-05-20]. Dostupné z: http://wiki.ros.org/Parameter%20Server

[31] ROS.org. *ROS Wiki: Messages* [online]. Open Source Robotics Foundation [cit. 2022-05-20]. Dostupné z: http://wiki.ros.org/Messages

[32] ROS.org. *ROS Wiki: actionlib_tutorials/Tutorials/SimpleActionServer* [online]. Open Source Robotics Foundation [cit. 2022-05-20]. Dostupné z: http://wiki.ros.org/actionlib_tutorials/Tutorials/SimpleActionServer%28ExecuteCallbackMethod%29

[33] ROS.org. *ROS Wiki: amcl* [online]. Open Source Robotics Foundation [cit. 2021-06-21]. Dostupné z: http://wiki.ros.org/amcl

[34] ROS.org. *ROS Wiki: move_base* [online]. Open Source Robotics Foundation [cit. 2021-06-21]. Dostupné z: http://wiki.ros.org/move_base

[35] Breach. *Bender robotics* [online]. [cit. 2022-05-20]. Dostupné z: https://www.benderrobotics.cz/breach.html

[36] *Stage* [online]. 2011 [cit. 2022-05-20]. Dostupné z: http://rtv.github.io/Stage/index.html

[37] Gazebo. *Gazebo* [online]. Open Robotics [cit. 2022-05-20]. Dostupné z: https://gazebosim.org/home

[38] Gazebo simulator. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2022 [cit. 2022-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Gazebo_simulator

[39] Breach. In: *Bender robotics* [online]. [cit. 2022-05-20]. Dostupné z: https://www.benderrobotics.cz/images/product_breach.jpg

[40] ROS.org. *ROS Wiki: actionlib* [online]. Open Source Robotics Foundation [cit. 2022-05-20]. Dostupné z: http://wiki.ros.org/actionlib