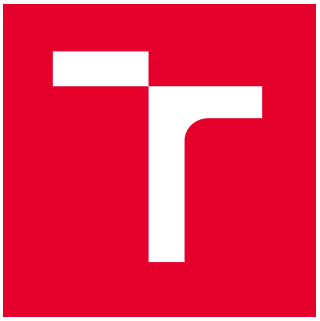


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

CHOVÁNÍ HEJNA MOBILNÍCH ROBOTŮ

SWARM BEHAVIOR OF MOBILE ROBOTS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jan David Šamánek**

AUTHOR

VEDOUCÍ PRÁCE — **mjr. Ing. Václav Křivánek, Ph.D.**

SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Bc. Jan David Šamánek**
Studijní program:	Mechatronika
Studijní obor:	bez specializace
Vedoucí práce:	**mjr. Ing. Václav Křivánek, Ph.D.**
Akademický rok:	2024/25

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

Chování hejna mobilních robotů

Stručná charakteristika problematiky úkolu:

Zkušenosti z použití robotu v dnešní době naznačují, že je výhodnější používáte konstrukčně jednodušší a tím i levnější roboty než drahé a komplikované solitéry. To ovšem vede k jinému přístupu, kdy se vytvářejí hejna robotů ne nepodobná hejnům ryb či ptáků. Není pak třeba dbát na spolehlivost jednoho jedince, za to se musíme zabývat chováním uvnitř hejna.
Práce se zabývá teorií chování hejna robotů na základní úrovni. Cílem práce je navrhnout vhodnou strukturu a vnitřní organizaci hejna robotů s ohledem na jejich schopnost vzájemné spolupráce.
Prvotním úkolem je realizace úloh přesunu hejna v různých formacích (konvoj, šíp, linie) nebo následně plnění koordinovaných úloh jako je např. prohledávání neznámého prostoru. Hejno je tvořeno několika autonomními jednotkami na základě robota DJI Robomaster S1. Pro ně je třeba navrhnout vhodnou HW nadstavbu tak, aby se mohli stát plnohodnotnými členy hejna.

Cíle diplomové práce:

1. Návrh metodiky a algoritmů pro řízení hejna robotů se zaměřením na úlohy formace, rozdělení úkolů a pokrytí prostoru.

2. Implementace vybraných algoritmů do robotického simulátoru a ověření základních vlastností.

3. Ověření výsledků na robotické platformě DJI Robomaster S1.

Seznam doporučené literatury:

[1] KOUBAA, A. Robot path planning and cooperation. New York: Springer, 2018.

[2] W. Bolton. Mechatronics: Electronic Control Systems in Mechanical and Electrical Engineering. 7th edition. 2018. ISBN 978-1292250977.

CORKE, P. Robotics, vision and control: fundamental algorithms in MATLAB. Cham: Springer, 2017

ABSTRAKT

V rámci diplomové práce je provedena důkladná rešerše, ve které jsou představeny hejnové algoritmy řešící problematiku agregace, prohledávání a pokrytí prostoru, rozdělení úkolů a další. Vybrané algoritmy jsou následně ověřeny na vytvořeném 2D simulátoru. Druhá část práce se zabývá vytvořením hardwarové nadstavby pro robota Robomaster S1 a následné ověření vybraného hejnového algoritmu na vytvořené robotické platformě.

KLÍČOVÁ SLOVA

ROS2, MicroROS, Docker, Jetson Nano, OpencR, Robomaster S1, hejnová robotika

ABSTRACT

As part of the thesis, a thorough literature review is conducted, presenting swarm algorithms that address the issues of aggregation, search and space coverage, task allocation, and more. The selected algorithms are then validated using a developed 2D simulator. The second part of the thesis focuses on creating a hardware extension for the Robomaster S1 robot and subsequently verifying the selected swarm algorithm on the developed robotic platform.

KEYWORDS

ROS2, MicroROS, Docker, Jetson Nano, OpencR, Robomaster S1, swarm robotics

ŠAMÁNEK, Jan David. *Chování hejna mobilních robotů*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky, 2025, 55 s. Diplomová práce. Vedoucí práce: mjr. Ing. Václav Křivánek, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Jan David Šamánek
VUT ID autora:	228743
Typ práce:	Diplomová práce
Akademický rok:	2024/25
Téma závěrečné práce:	Chování hejna mobilních robotů

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce mjr. Ing. Václavu Křivánku, Ph.D., který mi byl vždy, když jsem potřeboval, k dispozici. Dále bych rád poděkoval Bc. Filipu Drcmánkovi za zprostředkování svých 3D tiskáren a nakonec své rodině za podporu.

Obsah

Seznam obrázků

Úvod

Diplomová práce se zaměřuje na chování robotů v rámci hejna. Tento způsob kolektivního řízení skupiny robotů je inspirován přírodními systémy, jako jsou hejna ptáků či ryb, nebo kolonie mravenců, které dokážou dosáhnout komplexního chování a vysoké efektivity díky spolupráci a jednoduchým pravidlům interakce mezi jednotlivými členy.

Obr. 1: Spolupráce hejna mravenců

Hejnová robotika se zaměřuje na koordinaci skupin relativně jednoduchých robotů, kteří kooperují za účelem dosažení společného cíle. Výhodou tohoto přístupu je škálovatelnost, robustnost vůči poruchám jednotlivých členů a schopnost adaptace na měnící se prostředí.

Výše zmíněné vlastnosti robotických hejn přispěly k tomu, že v posledním desetiletí došlo k podstatnému nárůstu zájmu o tuto oblast výzkumu. Příkladem může být projekt *Swarm-bots* [3] financovaný Evropskou unií a projekt *Swarmanoid* [4], který navazuje na výsledky projektu *Swarm-bots*. Nebo soutěž *NASA Swarmathon* [1], která povzbuzovala soutěžící k inovacím v oblasti hejnové robotiky.

Není divu, hejnoví roboti jsou svými vlastnostmi skvělou volbou pro řadu misí. Ať už se jedná o mise v nehostinných podmínkách, kde hejno může těžit z postradatelnosti jedince. Příkladem může být meziplanetární explorace či hledání obětí havárie. Nebo mise, kde koordinace mezi jednotlivými roboty může zásadně přispět k rychlejšímu a efektivnějšímu dosažení cíle, jako například při těžbě v dolech nebo stavbě složitých konstrukcí. V takových scénářích může tým robotů, pracujících synchronizovaně, optimalizovat pracovní procesy tím, že rozdělí úkoly podle aktuální potřeby a kapacity jednotlivých členů.

I přes velký potenciál hejnových robotů a zvýšení zájmu o tuto problematiku ještě nedošlo k technologickému průlomu a hejnoví roboti zatím nenašli komerční využití.

Tato diplomová práce je rozdělena na teoretickou a praktickou část. V teoretické části jsou zkoumány různé algoritmy hejnových robotů se zaměřením na úlohy formace, prohledávání prostoru a rozdělení úkolů. Vybrané algoritmy jsou demonstrované na simulátoru vytvořeném v rámci této diplomové práce. Praktická část se zabývá návrhem vhodné hardwarové nadstavby pro robota *Robomaster S1* [2], tak aby mohl sloužit jako plnohodnotný člen hejna a aplikací vybraného algoritmu na hejno zhotovených robotů.

Obr. 2: Robomaster S1

Diplomová práce navazuje na bakalářskou práci Jana Bělohlávka [7], který rozklíčoval příkazy potřebné pro ovládání robota pomocí libovolného mikrokontroléru.

1 Rešerše - algoritmy

V následující kapitole budou rozebrány hejnové algoritmy se zaměřením na agregaci, prohledávání prostoru a sběr potravy, pokrytí prostoru, rozdělení úkolů a další. Součástí teoretické části bylo vytvoření 2D simulátoru, na kterém byly vybrané algoritmy implementovány.

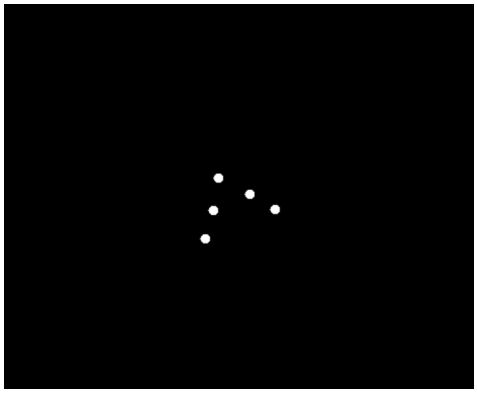
1.1 Simulátor

Simulátor byl implementován v jazyce C# za pomoci technologie WinForms. Simulátor implementuje i jednoduchou fyziku objektů, která umožňuje přidat do prostředí překážky. Tyto jednoduché překážky umožňují dokonce simulovat i poměrně komplikované úlohy.

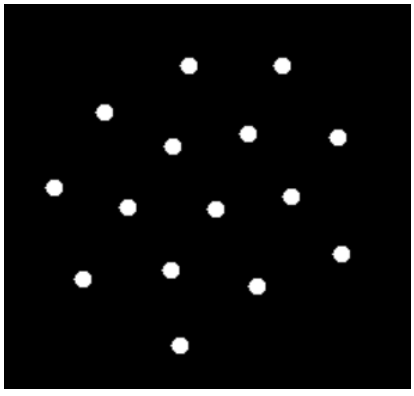
Simulace pracuje s lokální komunikací robotů. Každý robot je schopen komunikovat pouze do určité vzdálenosti, což prakticky simuluje reálné využití hejnových robotů ve ztížených podmínkách a zároveň přidává další omezení a výzvu pro algoritmus.

V rámci simulace je chování jednotlivých hejnových robotů simulováno modelem, který neuvažuje setrvačnost. Tato implementace vychází z provedené rešerše, kdy většina jednoduchých modelů robotických hejn žádnou setrvačnost neuvažuje a akčním zásahem je tedy rovnou požadovaná rychlost.

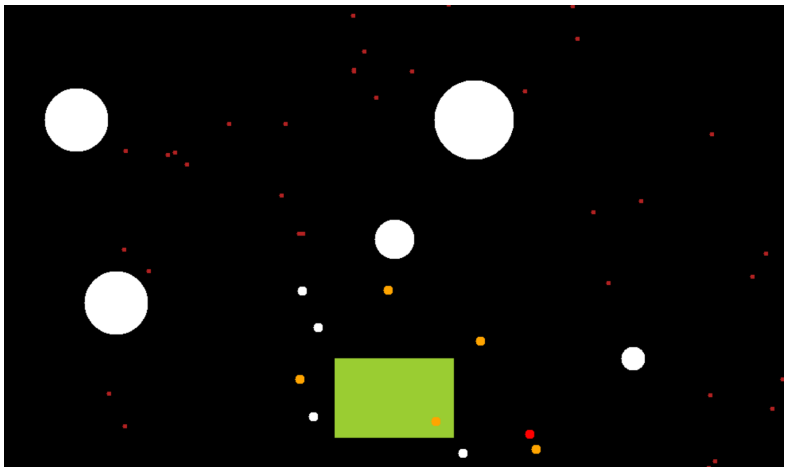
Na následujících obrázcích jsou ukázány některé z algoritmů, které byly implementovány v 2D simulátoru.



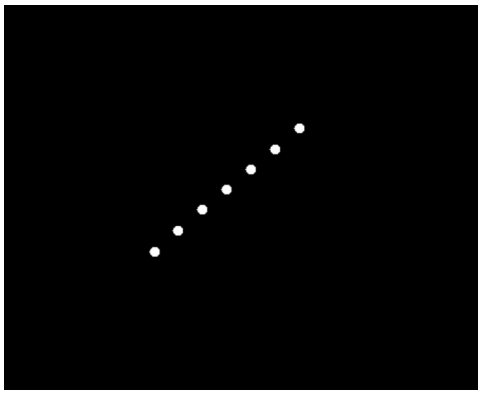
Obr. 1.1: Pohyb ve formaci šípu



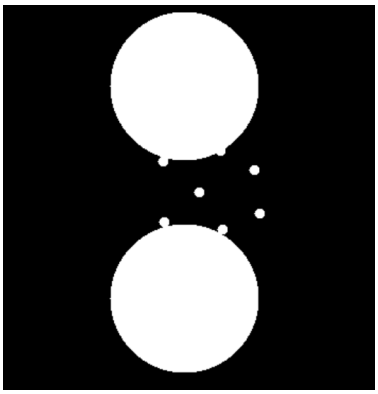
Obr. 1.2: Pokrytí prostoru



Obr. 1.3: Sběr potravy - CLW



Obr. 1.4: Uspořádání do line



Obr. 1.5: Pohyb ve formaci s překážkou

1.2 Agregace a pohyb ve formaci

Algoritmy zabývající se agregací a pohybem ve formaci se snaží navodit stav, při kterém se jednotliví jedinci hejna shromažďují a organizují do skupin nebo formací.

Tento proces je založen na jednoduchých pravidlech interakce mezi agenty, kde většina algoritmů vychází z myšlenky *umělého potencionálního pole*. Na každého člena hejna působí, na základě vzdáleností od ostatních jedinců, *virtuální síla*. Součet příspěvků všech virtuálních sil vytvoří umělé potenciální pole.

Virtuální síla má dvě hlavní složky: atraktivní sílu, která působí na větší vzdálenosti a podporuje sjednocení skupiny, a odpudivou sílu, která brání příliš těsnému shlukování nebo kolizím.

Algoritmy se pak liší především tím, jak virtuální sílu namodelujeme. Právě model virtuální síly určuje stabilitu a chování algoritmu.

Pro pohyb ve formaci je třeba vybrat jedince, který bude jednat jakožto vůdce hejna a bude určovat směr pohybu [5]. Pokud chceme dosáhnout specifického, slo-

žitějšího tvaru, je třeba namodelovat umělé potenciální pole na míru. To znamená určit jednotlivé virtuální síly mezi jedinci, neboli určit vzdálenosti jednotlivých jedinců. Praktickým příkladem je článek z roku 2018 [18], který využívá přístupu umělých potenciálních polí k pohybu hejna dronů ve formaci šípu.

V přírodě lze toto chování pozorovat například u ryb tvořících husté skupiny k zmatení predátorů nebo u mravenců, kteří se shromažďují při budování mostů z vlastních těl pro překonání překážek.

1.2.1 Model s měkkým potenciálem

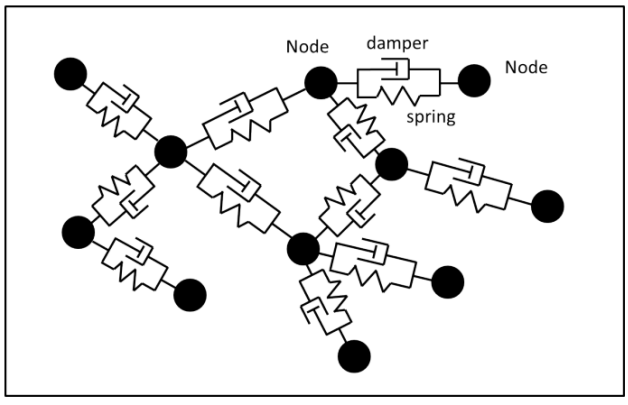
Model s měkkým potenciálem modeluje virtuální sílu pomocí následující rovnice.

$$u = -k \cdot \frac{d}{\|\vec{d}\|} \cdot \left(1 - \frac{d_s}{\|\vec{d}\|}\right) \tag{1.1}$$

$u =$ akční zásah
$k =$ koeficient síly odpudivého pole
$\vec{d} =$ vektor vzdálenosti mezi agenty
$\|\vec{d}\| =$ norma vektoru vzdálenosti
$d_s =$ mezní vzdálenost

1.2.2 Dynamický model

Tento model čerpá inspiraci ze zákonů dynamiky. Virtuální síly působící na jedince modeluje pomocí virtuální pružiny a tlumiče [16].



Obr. 1.6: Vizuální reprezentace dynamického modelu

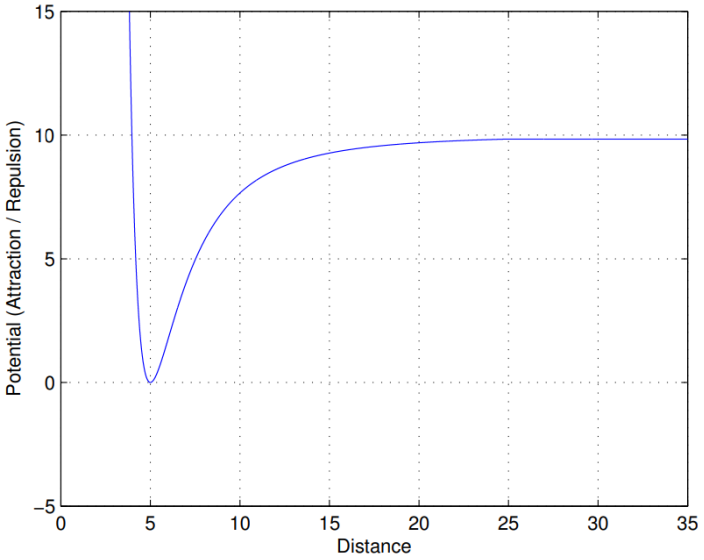
Každý jedinec je spojen se svými sousedy, nejbližšími jedinci hejna v jeho okolí, pomocí virtuální pružiny a tlumiče. Pro výpočet síly se použijí běžné dynamické

rovnice pro pružinu a tlumič, přičemž platí princip superpozice. Neboli platí, že výsledná síla bude odpovídat součtu vektorů všech virtuálních sil působících na jedince. Když známe sílu působící na jedince, je už jednoduché pomocí druhého Newtonova zákona spočítat zrychlení a následně rychlost, která je zároveň akčním zásahem.

K správnému fungování algoritmu je třeba vhodně zvolit tlumení virtuálního tlumiče a tuhost virtuální pružiny.

1.2.3 Lennard-Jonesův potenciál

Tento matematický model využívá k výpočtu virtuální síly Lennard-Jonesův potenciál [31]. Jedná se o potenciál, který se používá k výpočtu silového působení mezi dvěma molekulami. Správným nastavením konstant modelu jsme pak schopni navodit požadované chování hejna.



Obr. 1.7: Graf Lennard-Jonesova potenciálu

1.3 Pokrytí prostoru

Tato sekce diplomové práce se zaměří na rovnoměrné pokrytí prostoru, které může být využito například k vytvoření komunikační sítě, monitorování prostředí nebo k navádění ostatních robotů. Úloha pokrytí prostoru může být také využita k prohledávání prostoru, které je popsáno v sekci 2.3.

Jedná se o důležitou úlohu, která slouží u mnoha dalších úloh, například u pátrací a záchranné mise, jako přípravný krok.

V přírodě můžeme pozorovat úlohu obdobnou pokrytí prostoru například u mravenců, kteří při hledání potravy vytvoří komunikační síť a předávají instrukce o poloze potravy ostatním mravencům.

1.3.1 Algoritmus vektorů sil

Algoritmus silových vektorů se inspiroval způsobem, kterým se molekuly plynu rozšiřují, tak aby zaplnily nádobu. Molekuly plynu se pohybují z oblastí s vysokou hustotou do oblastí s nižší hustotou, tak dlouho, dokud se v rámci nádoby rovnoměrně nerozprostřou. Myšlenka, která stojí za algoritmem silových vektorů, je, že každý robot se chová jako molekula plynu, přičemž se pohybuje z oblastí s vyšší hustotou robotů do oblastí s jejich menším počtem. Tímto způsobem se hejno robotů rozprostře a pokryje oblast rovnoměrně [22].

Robot počítá výsledný vektor na základě myšlenky umělého potenciálního pole popsaného v předchozí kapitole. Virtuální síla se ale skládá pouze ze síly odpudivé. Robot se tedy pohybuje ve směru záporného gradientu hustoty pokrytí prostoru.

Pokud robot nemá v dosahu žádné další členy hejna a nemůže tedy určit svůj silový vektor, pokračuje v pohybu směrem naposled spočítaného silového vektoru, kdy se velikost vektoru postupem času postupně zmenšuje. Robot bude tímto způsobem dál pokračovat, dokud nepotká dalšího robota nebo se zcela nezastaví. To je užitečné, když se hejno snaží pokrýt rozsáhlou oblast.

1.3.2 Lloydův algoritmus

Lloydův algoritmus pro pokrytí prostoru vychází z teorie Voronoihových diagramů. Voronoihův diagram rozdělí prostor do tzv. Voronoihových buněk, vzhledem ke zvoleným bodů, přičemž každý bod v Voronoihově buňce má blíž ke zvolenému bodu uvnitř buňky než ke kterémukoli jinému zvolenému bodu.

Algoritmus zvolí jednotlivé členy hejna jakožto body Voronoihových buněk. Robot si na základě vzdáleností od svých sousedů a překážek v jeho okolí vytvoří Voronoihovu buňku. V dalším kroku spočítá pozici středu mu příslušné buňky a posune se směrem k tomuto středu. Vzdálenosti od překážek a sousedů se změní a celý proces se opakuje. Tímto iterativním procesem roboti postupně pokryjí daný prostor.

1.4 Prohledávání prostoru a sběr potravy

Prohledávání prostoru a sběr potravy jsou jedny ze stěžejních úloh pro hejnové roboty. Právě pro tento typ úloh jsou hejna vhodným řešením. Mohou těžit z na-

hraditelnosti a koordinace většího počtu jedinců.

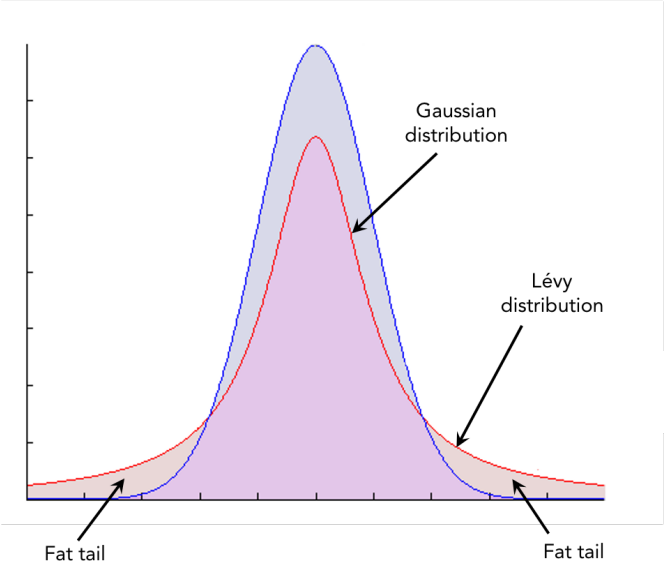
Praktickým příkladem může být nalezení a následná těžba ropných ložisek v nezmapovaném terénu, sběr materiálu na následnou stavbu nebo čistě zmapování neznámého prostředí.

Velká většina algoritmů využívá pravděpodobnostní stavové mašiny k řízení chování jednotlivých robotů. Stavy robota mohou být například prohledávání, těžba, odpočinek a další. Pravděpodobnost změny stavu ovlivňují různé faktory, jako třeba množství potravy nebo robotů v okolí. Na základě těchto faktorů upravují algoritmy pravděpodobnosti změn jednotlivých stavů. Stěžejní logika se skrývá v implementaci jednotlivých stavů a nastavení logiky pro změnu pravděpodobnosti přechodu do jiného stavu.

Zároveň je třeba vybrat vhodný algoritmus pro prohledávání prostoru.

1.4.1 Náhodný, Brownův a Léwyho pohyb

Náhodný, Brownův a Léwyho pohyb jsou tři základní strategie pro prohledávání prostoru. Každý robot prohledává prostor tzv. kroky. Každý krok je učiněn v náhodném směru s náhodnou délkou. O jaký typ pohybu se jedná, určuje distribuce, z které je délka vybírána,



Obr. 1.8: Porovnání Léwyho a normálního rozdělení

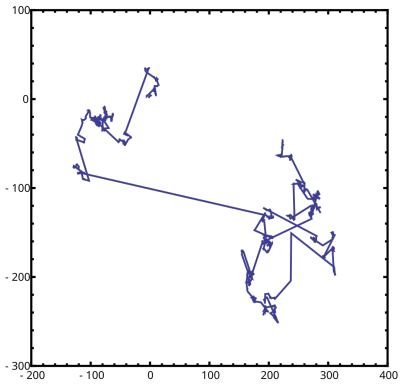
Náhodný pohyb je nejjednodušší variantou prohledávání pohybu. Délku vybírá většinou z rovnoměrného rozdělení nebo má dokonce délku kroku danou.

Brownův pohyb volí délku na základě normálního rozdělení. Je vhodný pro prohledávání prostoru, kde jsou cíle koncentrované blízko u sebe. Vychází z náhodného

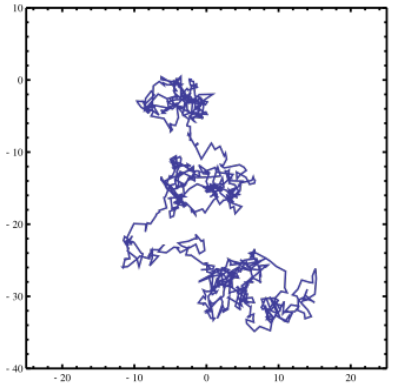
pohybu mikroskopických částic v kapalném nebo plynném médiu, které konají právě Brownův pohyb.

Pro prohledávání prostoru je nejvhodnější Léwyho pohyb. Označuje pohyb, který délku kroku volí z Léwyho distribuce. Léwyho distribuce pravděpodobnosti je distribuce, která má tzv. fat tail. Což má za důsledek, že výskyt velmi velkých hodnot neklesá tak rychle jako třeba u normálního rozdělení. To je vidět na následujícím obrázku.

Léwyho pohyb je považován za nejlepší strategii pro náhodné prohledávání prostoru. Vyskytuje se i v přírodě, kde stejný pohyb praktikují mravenci při hledání potravy.



Obr. 1.9: Léwyho pohyb



Obr. 1.10: Brownův pohyb

1.4.2 Algoritmy CLW a ACLW

Základem pro algoritmy CLW (*Collective Lewy Walk*) a ACLW (*Adaptive Collective Lewy Walk*) je Léwyho pohyb, přičemž každý z nich je navržen k řešení odlišných problémů spojených s prohledáváním prostoru [21].

Za předpokladu, že velké množství robotů začne s prohledáváním ze společného stanoviště, je logické, že dojde k mnoha srážkám a tím ke znekvalitnění, zpomalení procesu prohledávání.

Tento problém se snaží řešit algoritmus CLW. Navrhuje metodiku, kdy spolu roboti komunikují a vysílají informaci o tom, zda provádějí tzv. dlouhý let. Za dlouhý let označujeme krok s délkou větší než zvolená délka l. Roboti, kteří neprovádí dlouhý let, jsou odpuzováni pomocí umělého potenciálního pole od robotů, co dlouhý let provádí. Robotům s dlouhým letem je tedy dána priorita a tím se zajistí, že se rychleji prohledá celý prostor.

Často se stává, že se potrava vyskytuje koncentrovaná ve shlucích. Pokud robot narazí na potravu, může předpokládat, že se v blízkém okolí nachází další. V

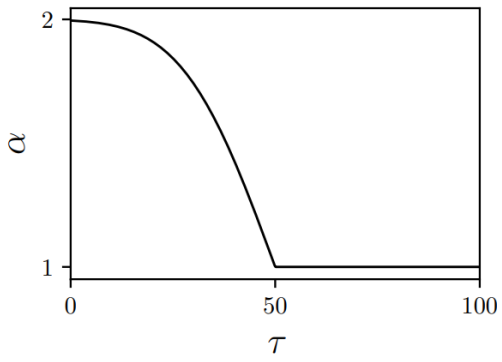
takovém případě je pro hledání potravy vhodnější Brownův pohyb, který efektivněji hledá cíle koncentrované blízko u sebe. Algoritmus ACLW využívá vlastností Léwyho distribuce ke změně typu pohybu na základě doby uplynulé od posledního nálezu potravy.

Hodnotu odpovídající Léwyho rozložení pravděpodobnosti získáme pomocí Mantegnaova algoritmu. Kde u, v jsou hodnoty získané z normálního rozdělení a α je parametr určující pravděpodobnost výskytu velkých hodnot, velikost tzv. fat tailu. Pokud zavedeme odpovídající normalizaci, konverguje daná hodnota z_n pro $n > 100$ k Léwyho distribuci.

$$m = \frac{u}{|v|^{1/\alpha}} \qquad z_n = \frac{1}{n^\alpha} \sum_{k=1}^{n} m_k \tag{1.2}$$

Bude-li parametr α roven 1, generuje distribuce Brownův pohyb. Naopak když je parametr α roven 2, generuje distribuce Léwyho pohyb. Přechod mezi hodnotou 1 a 2 tvoří plynulý přechod mezi Brownovým a Léwyho pohybem.

Téhle skutečnosti využívá algoritmus ACLW, který upravuje parametr α. Ten mění pohyb z Léwyho na Brownův na základě uplynutí doby od posledního nalezení potravy podle funkce $\alpha = \max(1, \operatorname{erfc}[\beta(\tau - C)])$ zobrazené na následujícím grafu.



Obr. 1.11: Průběh parametru α po nalezení potravy

Tím se dosáhne chování, kdy robot při nalezení potravy hledá v blízkém okolí po další potravě a po čase znova přepne na Léwyho pohyb a koná efektivnější průzkum prostoru.

1.4.3 Algoritmus firefly

Algoritmus se inspiroval chováním světlušek, konkrétně jejich bioluminiscencí. Světlušky využívají své světlo k přilákání partnerů. Stejně jako předchozí algoritmy používá algoritmus firelfy k prohledávání prostoru Léwyho pohyb [32].

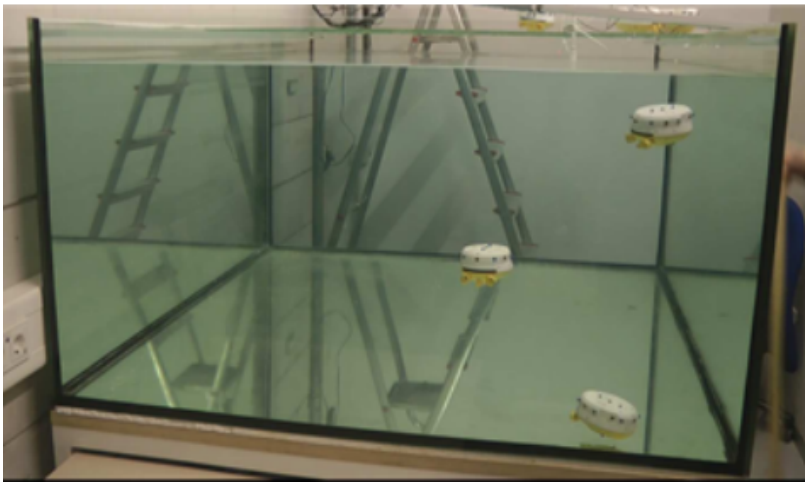
Tento algoritmus se snaží řešit stejný problém jako algoritmus ACLW popsaný v předchozí kapitole. Pokud se v daném prostoru nachází potrava, lze předpokládat, že v blízkém okolí bude koncentrováno více potravy než v jiném nahodilém místě. Algoritmus firefly řeší daný problém tak, že pomocí lokální komunikace přiláká do této oblasti více robotů.

Každý robot si udržuje v paměti hodnotu atraktivity, která je dána dobou uplynulou od posledního nalezení potravy a počtem nalezené potravy. Pokud tedy robot nalezne potravu, zvýší hodnotu své atraktivity, jinak jeho atraktivita lineárně klesá viz. následující rovnice.

$$x_i(n+1) = \hat{x}_i(n) + \beta_0 e^{-\mu r_{ij}^2}(\hat{x}_j - \hat{x}_i) + b(\text{rand} - 0.5) \tag{1.3}$$

$$\begin{aligned}
\hat{x}_i &= \text{pozice robota } i, \\
\hat{x}_j &= \text{pozice robota } j, \\
\beta_0 &= \text{atraktivita robota } j, \\
\mu &= \text{koeficient absorpce atraktivity}, \\
r_{ij} &= \text{vzdálenost mezi roboty } i \text{ a } j, \\
b &= \text{váhový koeficient [0:1]}.
\end{aligned}$$

Při prohledávání prostoru Léwyho pohybem komunikuje robot s ostatními roboty v jeho okolí, s každým robotem porovnává svou hodnotu atraktivity a následně upraví svou dráhu směrem k robotovi s vyšší atraktivitou. Tímto způsobem jsou roboti lákáni k oblasti zájmu.



Obr. 1.12: Ukázka z reálného experimentu algoritmu firefly

1.5 Rozdělení úkolů

Rozdělení úkolů je klíčová úloha v oblasti hejnové robotiky, která získala v posledních 20-ti letech velkou pozornost.

Řeší problematiku toho, jak zajistit, aby roboti efektivně využívali dostupné zdroje a plnili úkoly v co možná nejkratším čase. Tohoto chování mohou hejnové systémy dosáhnout vhodným rozdělením úkolů nebo patřičným navázáním spolupráce mezi roboty. Bez správného rozdělení úkolů mohou někteří roboti zůstat nečinní, zatímco jiní jsou přetížení.

Je třeba brát v potaz, že hejnové systémy jsou ze své povahy decentralizované, tudíž každý robot se rozhoduje sám, což činí řešení podstatně složitějším.

1.5.1 Rozdělení úkolů na základě modelu prahové reakce

Algoritmus řeší problém rozdělení práce na modelovém příkladu sběru potravy.

Roboti mají společné hnízdo, kde je potrava spotřebovávána, každý robot se sám rozhoduje, zda je třeba začít hledat další potravu nebo odpočívat, a to bez jakékoli komunikace mezi roboty, čistě z lokálního pozorování [34].

Tato úloha může simulovat stavbu konstrukce, kde jedna část robotů staví konstrukci a druhá hledá materiál pro stavbu nebo případ, kdy je třeba vždy dobíjet baterie robotů, aby se nevybili, zatímco druhá část hledá prostředky v okolí.

Algoritmus je postaven na myšlence modelu prahové reakce, což je matematický model, který simuluje rozdělení práce u hmyzu.

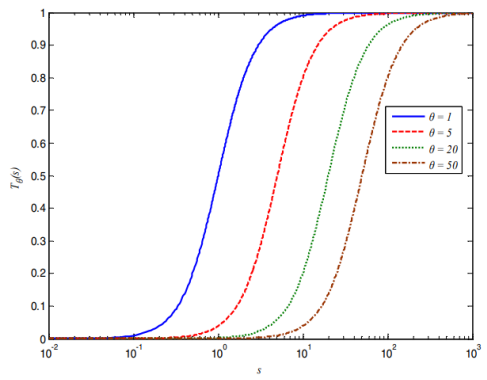
Matematická reprezentace modelu je znázorněna následující rovnicí.

$$T_\theta(s) = \frac{s^n}{s^n + \theta^n} \tag{1.4}$$

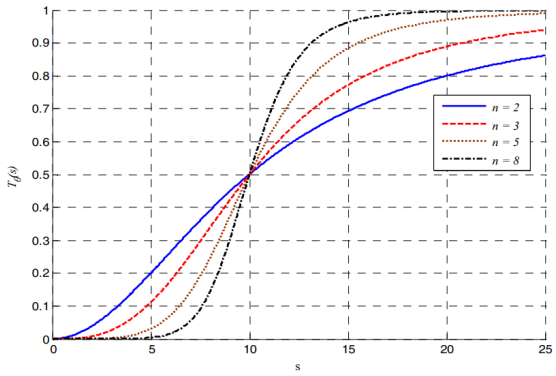
Kde s znázorňuje stimul související s danou úlohou, kterým může být například počet setkání s ostatními roboty, chemická koncentrace určité látky v okolí nebo jakýkoli jiný indikátor. Proměnná θ znázorňuje práh. Graf modelu pro různé θ a různé n je znázorněn na následujících obrázcích.

Na začátku jsou všichni roboti inicializováni se stejným prahem θ, ale jinými hodnotami n. Jiné hodnoty n mění náklon funkce $T_\theta(s)$ a tudíž pravděpodobnost reakce na daný stimul, to zajistí, že každý jedinec bude na stimul reagovat jinak, což povede k větší robustnosti algoritmu.

Každý robot se rozhoduje na základě modelu prahové reakce $T_\theta(s)$ zda zůstane v hnízdu nebo se vydá hledat potravu. To činí na základě pozorování stavu v hnízdě, zda je v hnízdě dostatek potravy nebo třeba materiálu pro stavbu.



Obr. 1.13: Graf prahových reakcí s různými prahovými hodnotami



Obr. 1.14: Graf prahových reakcí s různými n

1.5.2 Rozdělení sekvenčně závislých úkolů

Mnoho úloh v reálném světě je na sobě závislých, je třeba nejdříve vykonat jednu úlohu a až pak je možné vykonat úlohu následující. Pokud nebude první úloha vykonána, nelze pokračovat v úloze další.

Aby byla práce na celkové úloze co nejefektivnější, je třeba rozdělit roboty mezi úlohy, tak aby se minimalizoval celkový čas plnění.

Příkladem může být úloha, kdy jedna část robotů těží určitý materiál, který následně složí na odkladišti, kde materiál převezme druhá část robotů a odveze ho do skladu. Pokud nebude dostatek materiálu pro odvoz do skladiště, měl by se robot autonomně rozhodnout, že půjde pomoci s těžbou, a naopak pokud se bude hromadit materiál na odkladišti, je třeba přeřadit část robotů z těžby na odvoz.

Jedno možné řešení nabízí skupina z Bruselské univerzity [6]. Skupina navrhla algoritmus pro rozdělení úkolů, kdy roboti interagují na daném rozhraní, které může představovat třeba odkladiště pro materiál.

Metoda je založena na měření *zpoždění na rozhraní* (d_{ij}), tedy času, který robot tráví čekáním na interakci s robotem z druhé skupiny. Každý robot si udržuje svůj odhad průměrného zpoždění pro oba směry interakce:

- $\hat{d}_{ij}$: Průměrné zpoždění robotů pracujících na aktuálním úkolu, čekajících na interakci s roboty z druhého úkolu τ_j.
- $\hat{d}_{ji}$: Průměrné zpoždění robotů z druhého úkolu, čekajících na roboty z úkolu prvního.

Průměrné zpoždění $\hat{d}_{ij}$ se aktualizuje pokaždé, když robot skončí s čekáním na interakci pomocí váženého průměru:

$$\hat{d}_{ij} \leftarrow 0.2 \cdot d_{ij} + 0.8 \cdot \hat{d}_{ij}, \tag{1.5}$$

Na základě těchto odhadů každý robot rozhoduje, zda je třeba ho přeřadit k jinému úkolu, pomocí pravděpodobnostní funkce. Pravděpodobnost přechodu robota

z jednoho úkolu na druhý je definována pomocí funkce sigmoid jako:

$$p_{ij}(d_{ij}) = \frac{1}{1 + e^{-\theta(d_{ij})}} \cdot \gamma, \tag{1.6}$$

kde γ je parametr škálování a $\theta(d_{ij})$ je následující funkce:

$$\theta(d_{ij}) = \frac{1}{k}\left(\frac{d_{ij}}{r(\hat{d}_{ij}, \hat{d}_{ji})} - m\right). \tag{1.7}$$

Parametr k určuje, jak prudce se pravděpodobnost p_{ij} mění, zatímco parametr m nastavuje, při jaké hodnotě zpoždění se přepínání začne uvažovat. Funkce $r(\hat{d}_{ij}, \hat{d}_{ji})$ zajišťuje, že rozhodování je asymetrické a závisí na poměru mezi průměrnými zpožděními:

$$r(\hat{d}_{ij}, \hat{d}_{ji}) = \begin{cases} \frac{\hat{d}_{ij}^2}{\hat{d}_{ji}}, & \text{pokud } \hat{d}_{ij} > \hat{d}_{ji}, \\ \hat{d}_{ij}, & \text{jinak.} \end{cases} \tag{1.8}$$

Roboti na rozhraní měří svá zpoždění a rozhodují se na základě výše uvedené pravděpodobnostní funkce. Pokud se robot z jedné skupiny přeřadí do druhé skupiny, zlepší tím rozdělení pracovních sil a sníží zpoždění na rozhraní.

Výsledkem je, že zpoždění na rozhraní se pro jednotlivé skupiny robotů vyrovná a úkoly jsou zpracovány efektivně. Roboti jsou tímto způsobem schopni bez přímé komunikace dosáhnout téměř optimálního rozdělení mezi dvě, potenciálně i více skupin.

1.6 Další

Za zmínku ještě určitě stojí algoritmus pro detekci poruchy člena hejna [8], který se stejně jako algoritmus *Firefly* inspiroval chováním světlušek. Používá princip prahového modelu popsaného v předchozích kapitolách k synchronizaci pomocí světelných signálů. Roboti se nejdříve synchronizují (všichni roboti blikají ve stejný moment). Následně každý robot detekuje ostatní roboty, kteří nejsou synchronizováni. To může znamenat, že blikají jinou rychlostí nebo neblikají vůbec. Tímto způsobem detekují poruchu jiného robota.

Stejně jako ve všech ostatních oblastech, i v hejnové robotice našla své místo umělá inteligence. Velkým krokem vpřed pro použití umělé inteligence u robotických hejn udělala skupinka vědců, mezi kterými byli i čeští vědci z ČVUT [20]. Ti navrhli decentralizovaný způsob učení *FLDDPG*. Tento způsob učení je navržen zejména pro prostředí s omezenou možností komunikace, jako jsou podzemní prostory nebo podvodní prostředí.

Algoritmus FLDDPG využívá federované učení k decentralizovanému tréninku jednotlivých robotů. Každý robot trénuje lokální neurální síť na základě vlastních

dat ze senzorů, která nejsou sdílena s žádným centrálním serverem. Váhy lokálních modelů jsou ovšem jednou za čas poslány na centrální server a aktualizovány, což umožňuje, aby se roboti nepřímo učili ze zkušeností ostatních, aniž by si sdíleli surová data. Tento přístup minimalizuje komunikační nároky a zachovává přesnost modelů.

Dalším zajímavým příkladem využití umělé inteligence v hejnové robotice je rozpoznání robota, který se snaží infiltrovat hejno a negativně ovlivnit jeho chování [33]. Nepřátelský robot v hejnu je detekován pomocí natrénované rekurentní neuronové sítě LSTM. Metoda byla validována na simulovaných datech, přičemž model dosáhl více než 90% přesnosti.

2 Rešerše - Hardware

Tato část představí již vytvořené hejnové roboty, kteří mohou sloužit jako inspirace pro hardwarovou nadstavbu.

2.1 Swarm-bot a s-bot

S-bot je robot vyvinutý skupinou Francesco Mondada v Laboratoire de Systme Robotiques (LSRO) z EPFL [9]. Jedná se o plně autonomního robota, který je schopný plnit jednoduché úkoly, jako je autonomní navigace, vnímání svého okolí a uchopení předmětů. Navíc je s-bot schopen komunikovat s ostatními s-boty a dokonce se na ně i fyzicky připojit. Pokud se spojí více robotů, vytvoří komplexnějšího robota nazývaného *swarm-bot*. Swarm-bot je tedy robot poskládaný ze spojených s-botů.

Obr. 2.1: S-bot

Obr. 2.2: Swarm-bot

Robotická platforma swarm-bot byla speciálně navržena pro úkoly, kde je třeba spolupráce mezi roboty, jako je třeba transport těžkého materiálu nebo překonání překážky bránící robotům v přesunu.

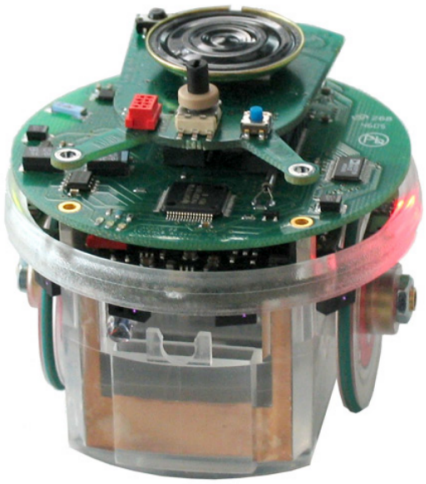
S-bot má diferenciální podvozek, nad kterým je věž s řadou senzorů a dva uchopovače pro navazování spojení s jinými roboty. Je vybaven senzory potřebnými pro navigaci, jako jsou infračervené senzory blízkosti, senzory pro detekci a rozeznání světla, akcelerometry a inkrementální enkodéry. Detekci a komunikaci s ostatními roboty řeší s-bot díky všesměrové kameře, LED diodám kolem věže a zvukovým vysílačům a přijímačům. Každý kloub robota má dále senzory pro měření momentu/síly.

2.2 E-puck

Jedná se o robota vyvinutého pro vzdělávací účely Michaelem Bonanim a Francescem Mondadem [14]. E-puck je malý (7 cm v průměru) robot s diferenciálním řízením dvou krokových motorů.

Je vybaven 16-bitovým mikrokontrolérem Microchip dsPIC s rychlostí 64 MHz. Svět kolem sebe vnímá *e-puck* díky 8 infračerveným senzorům rozmístěných po obvodu, 3D akcelerometru, třem mikrofonům pro zachycení zvuku a barevné CMOS kameře.

S ostatními roboty interaguje pomocí reproduktoru, osmi červených LED diod rozmístěných po obvodu, sadě zelených LED světel a technologie Bluetooth.



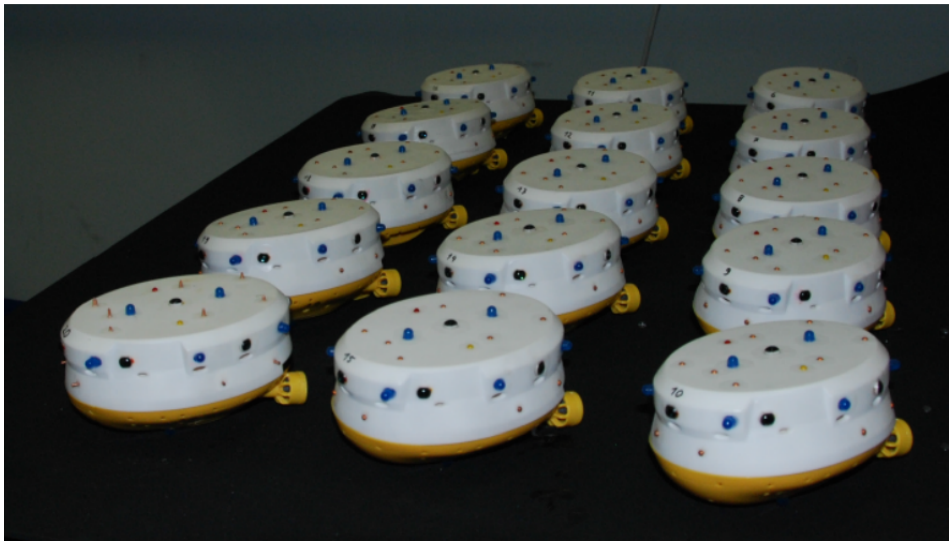
Obr. 2.3: E-puck

Robot e-puck je zcela *open source* a to jak hardware, tak software. To umožňuje širokou přizpůsobitelnost a vedlo k velkému počtu nadstaveb vytvořených pro robota e-puck. Příkladem může být rozšíření o Zigbee komunikaci nebo rozšiřující Pi-puck deska umožňující propojení s Raspberry Pi.

2.3 Lily

Robotická platforma *Lily* byla vytvořena v rámci projektu CoCoRo financovaného Evropskou unií [32]. Jedná se o podvodního robota, který se pohybuje ve vertikálním směru za pomoci dvou vrtulí a horizontálně díky vztlakovému čerpadlu.

Lily využívá senzor modrého světla a LED diody k měření vzdálenosti mezi roboty a popřípadě i detekci překážek. Senzor modrého světla slouží také ke komunikaci, kdy Lily dokáže komunikovat s ostatními roboty rychlostí až 100 kbps.



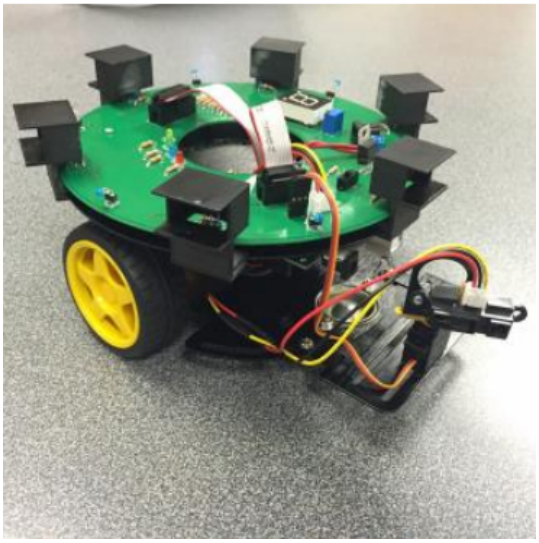
Obr. 2.4: Lily

2.4 Antz robot

Antz robot je jednoduchý robot vytvořený pro ověření výsledků práce zabývající se algoritmem vektorů sil (viz kapitola 2.2.1).

Hlavní výpočetní sílou Antz robota je Arduino MEGA. Robot má 6 infračervených senzorů rozložených ekvidistantně po obvodu. V přední části robota je vestavěn senzor pro měření vzdálenosti.

Antz robot se velkou mírou inspiroval od robota E-puck.



Obr. 2.5: Antz robot

3 Hardware

Praktická část této práce se bude zabývat vytvořením vhodné hardwarové nadstavby pro robota Robomaster S1 viz. Obr. 2. Zaměří se na hardware vybraný pro hejnového robota vytvořeného v rámci diplomové práce.

Jak již bylo řečeno v úvodu, tato diplomová práce navazuje na výsledky bakalářské práce Jana Bělohlávka [7]. Ten umožnil ovládání pohybu robota Robomaster S1 pomocí příkazů pravidelně posílaných přes sběrnici CAN.

Robot vytvořený v rámci této práce má později sloužit i k úlohám podstatně výpočetně náročnějším než je úloha pohybu ve formaci. Při výběru hardwaru byl tento fakt brán v potaz.

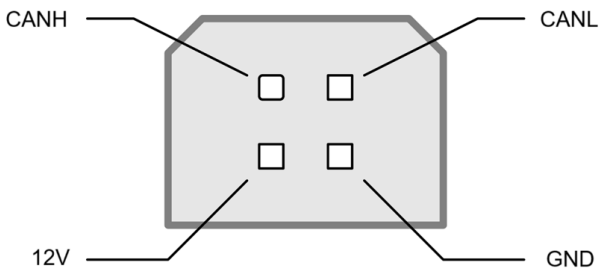
Celý hardware design byl inspirován robotem TurtleBot3 [27] a je nadstavbou nad robotem Robomaster S1.

3.1 Robomaster S1

Jedná se o vzdělávacího robota vyvinutého společností DJI, který je využíván i v řadě robotických soutěží. Robomaster S1 nabízí řadu funkcí založených na umělé inteligenci, jako je rozpoznávání obličejů, gest nebo objektů.

K robotovi je vytvořena aplikace, která umožňuje programovat robota v jazyce Scratch a Python. Tento způsob programování je částečně nahrazen prací Jana Bělohlávka [7].

Je osazen všesměrovými koly, které mu umožňují pohybovat se v libovolném směru nezávisle na směru natočení. Robota pohání motor M3508I, který je schopen dosáhnout maximálních otáček: 1000 ot./min a vyvinout maximální moment: 0.25 Nm. Robot je schopen vyvinout rychlost až 3.5 m/s.



Obr. 3.1: Rozložení pinů na robomaster S1 konektoru

Disponuje věží, která slouží jako střílna, může střílet buď plastové kuličky, nebo infračervený paprsek. Zásah detekuje robot pomocí senzorů umístěných na všech

stranách podvozku. Na této věži se mimo jiné vyskytuje kamera a inteligentní kontrolér, který bude nahrazen následujícím hardwarem.

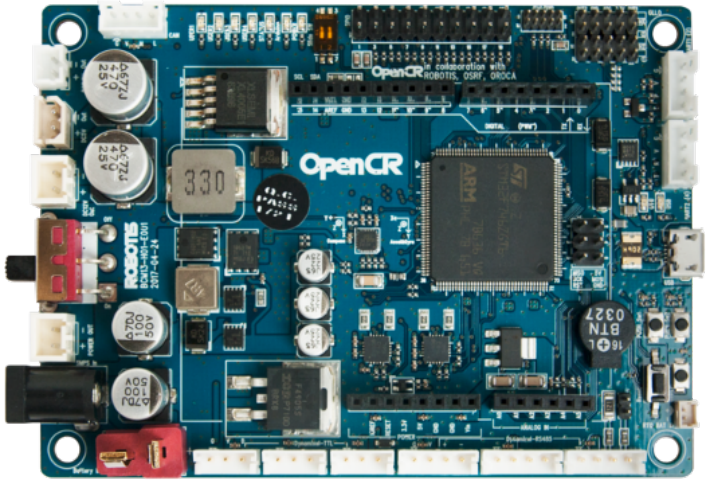
3.2 Deska OpenCR

K posílání CAN příkazů je využita deska OpenCR [26], která má v sobě zabudovaný vysílač pro posílání CAN zpráv a není tedy pro zadání příkazů žádného dalšího externího hardwaru.

Jedná se o open-source vývojovou desku vyvinutou speciálně pro roboty, kteří využívají ROS, open-source framework pro vývoj softwaru v robotice (více v pozdějších kapitolách zabývajících se softwarem).

Hlavním procesorem desky je výkonný mikrokontrolér STM32F7, který zajišťuje nízkou latenci a tedy real-time zpracování dat. Deska OpenCR podporuje USB, UART, I2C, SPI, TTL, RS485 a CAN pro komunikaci s externími zařízeními, k desce lze tedy připojit velkou řadu senzorů. Disponuje mnoha PWM, ADC převodníky a GPIO vstupy/výstupy. Má v sobě již zabudovaný gyroskop a akcelerometr a je schopna na výstupu vytvořit napětí 12, 5 a 3.3 voltů.

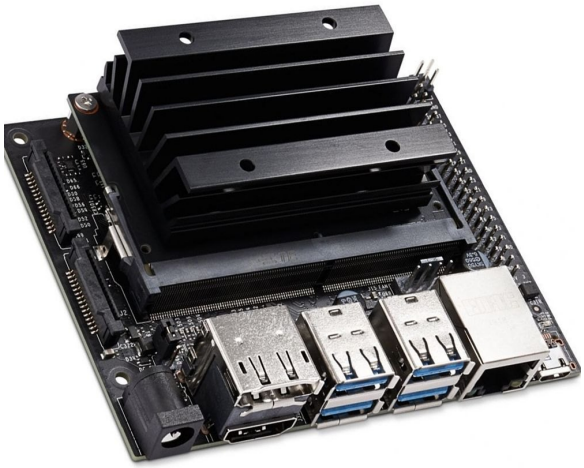
Navíc podporuje programování pomocí prostředí Arduino, což umožňuje využívat známé knihovny a jednoduchý kód pro řízení robotů. To činí OpenCR přístupnou pro začátečníky i pokročilé vývojáře.



Obr. 3.2: Deska OpenCR

3.3 NVIDIA Jetson Nano

Jako mozek robota byla zvolena vysoce výkonná vývojová deska NVIDIA Jetson Nano, která je určena pro vývoj umělé inteligence mobilních robotů [30].



Obr. 3.3: NVIDIA Jetson Nano Developer Kit

Disponuje čtyřjádrovým procesorem ARM Cortex-A57 a 128-jádrovým GPU a 4GB LPDDR4 pamětí, což umožňuje efektivní zpracování obrazů, strojové učení a další náročné výpočty. Disponuje dvěma vstupy, na které je možné připojit kameru. Komunikaci zajišťují sběrnice: UART, SPI, I2S, I2C. K desce lze dokonce připojit Ethernet nebo HDMI kabel.

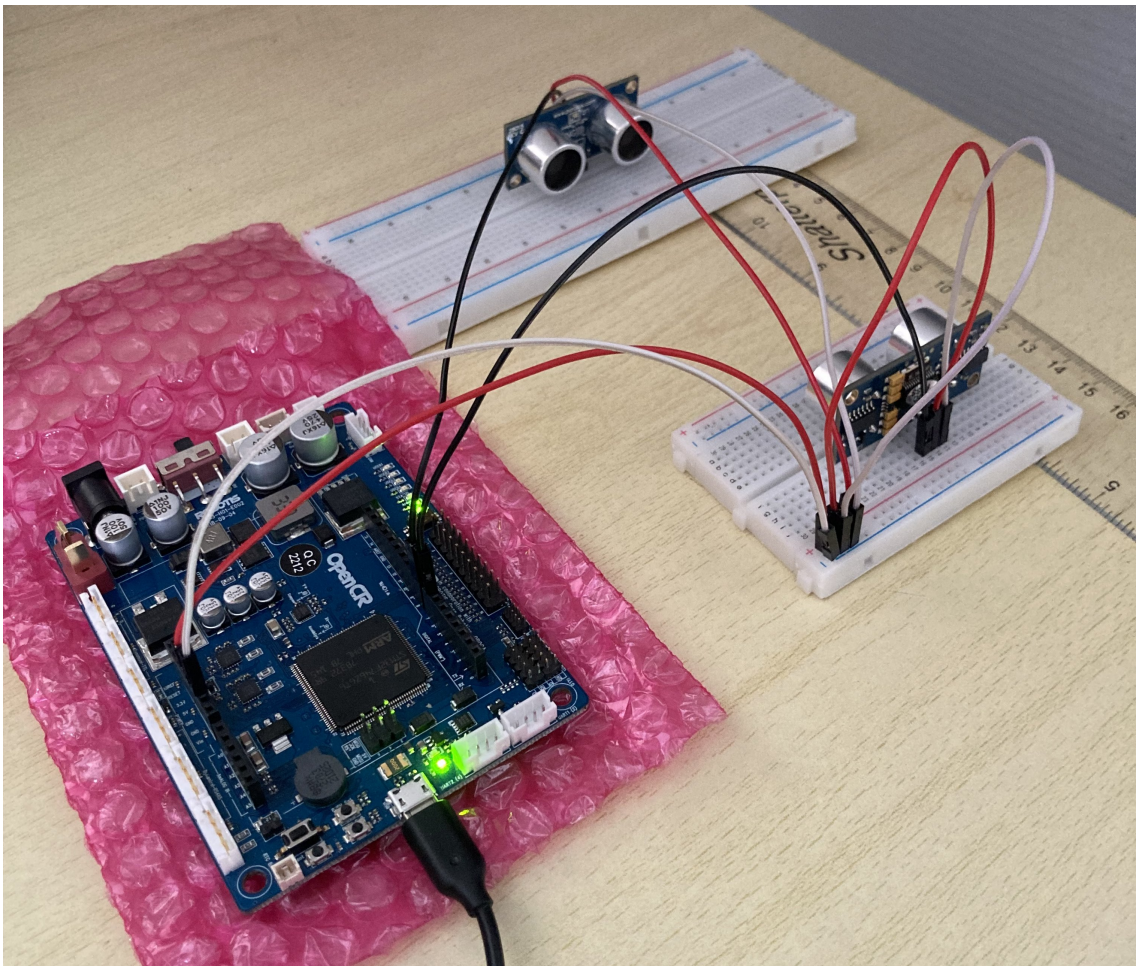
Jetson se využívá k úlohám, které vyžadují vysoký výkon. Deska OpenCR by tyto úlohy nemusela nezvládnout kvůli omezené výpočetní kapacitě.

3.4 RPLiDAR

Pro implementaci úlohy pohybu ve formaci je třeba měřit vzdálenost od jednotlivých členů hejna. Prvotní myšlenka byla rozmístit po obvodu robota více ultrazvukových senzorů pro měření vzdálenosti. Ale ukázalo se, že by mohl být problém s přeslechem ultrazvuků.

Ultrazvukový senzor měření vzdálenosti funguje na principu měření doby letu zvukového signálu. Senzor vyšle signál a čeká, než se vrátí, následně na základě uplynulého času odhadne vzdálenost. Pokud se ovšem v blízkosti nachází další ultrazvukový senzor, může se stát, že senzor nezachytí svůj zvukový signál, ale zvukový signál, který mu nepřísluší, dojde k přeslechu. Tato skutečnost byla ověřena na

jednoduchém experimentu.



Obr. 3.4: Test přeslechu ultrazvukových senzorů Parallax 28015

Pro měření vzdálenosti mezi roboty byl nakonec vybrán RPLiDAR. LiDAR funguje na podobném principu jako ultrazvukový senzor vzdálenosti. Vysílá laserový paprsek, čeká na odražený paprsek a na základě uplynulé doby mezi vysláním a přijetím počítá vzdálenost. Stejně jako u ultrazvukového senzoru může tedy dojít k přeslechu, kdy LiDAR bude přijímat paprsek jiného LiDARu. LiDAR ale vysílá pouze paprsek, zatímco ultrazvukový senzor vysílá zvukovou vlnu, pravděpodobnost interference je tak podstatně menší.

Na rozdíl od ultrazvukového senzoru, kde máme k dispozici pouze jednu hodnotu vzdálenosti, LiDAR poskytne hodnoty z celého okolí. Postupně se otáčí a snímá vzdálenost ve všech směrech. Mimo informace o vzdálenosti je k dispozici ještě informace o intenzitě odraženého paprsku.

Díky většímu množství změřených vzdáleností a informacím o intenzitě odraženého paprsku je možné případné interference vyfiltrovat a tím vyřešit případný problém chybného měření vzdálenosti.

U LiDARu RPLiDAR je možnost upravit rychlost otáčení motoru, neboli frekvenci snímání okolí. Nastavením jiné frekvence snímání okolí u každého robota dále snížíme pravděpodobnost interference paprsku.

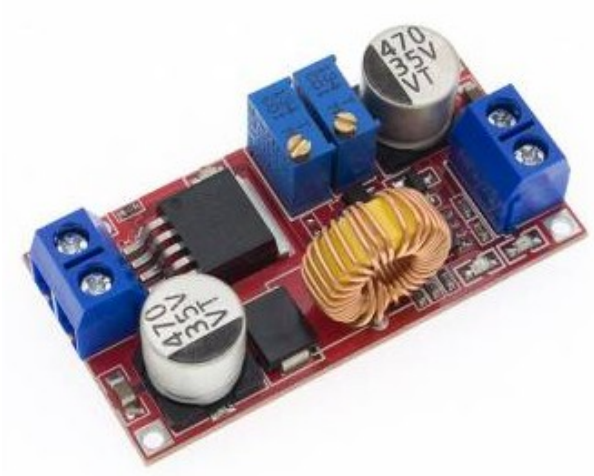
Obr. 3.5: RPLiDAR A3

3.5 DC/DC konvertor XL4015

Jetson Nano má napájecí napětí 5 V a odebírá maximální proud o velikosti 4 A. Robomaster S1 zpřístupňuje napětí 12 V.

Pro snížení napětí na 5 V byl vybrán DC/DC konvertor XL4015 [10], který má rozsah vstupního napětí od 4 V do 30 V a rozsah výstupního napětí od 1.25 V do 28 V a dokáže dodat maximální proud až 4 A.

Je tedy vhodným konvertorem pro napájení Jetsonu Nano.



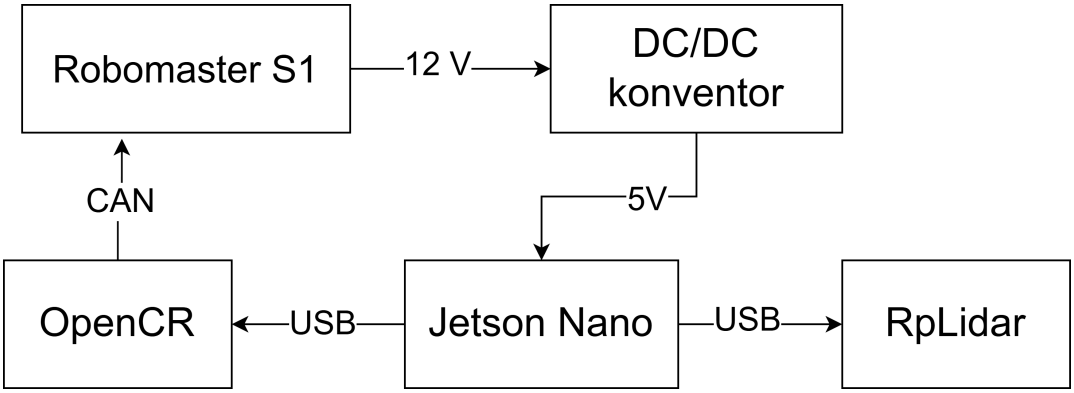
Obr. 3.6: DC/DC konvertor XL4015

3.6 Kompletace

Pro spojení jednotlivých elektronických komponent byly vytvořeny odpovídající konektory.

Jetson Nano je napájen pomocí DC/DC konvertoru, připojeného na 12V výstup Robomastera S1. Jetson Nano dále napájí pomocí USB kabelů RPLidar a desku OpenCR.

Zapojení elektronických komponent je přehledně znázorněno na následujícím diagramu.



Obr. 3.7: Hardware zapojení - přehled

Aby bylo dostatek prostoru pro instalaci veškerého hardwaru na robota Robomaster S1, bylo třeba odmontovat věž/střílnu a vytvořit rám pro elektroniku.

Za pomocí rozměrů z fotografie původního uchycení věže byla vytvořena komponenta, která přesně odpovídá montážním bodům původní věže a slouží jako instalační intermezzo pro zbytek vytvořeného rámu.

Pro tento mezistupeň byly navrženy a vytvořeny jednotlivé stupně, které umožňují snadné vzájemné propojení. Tyto jednotlivé úrovně lze jednoduše skládat na sebe, čímž vzniká flexibilní konstrukce pro organizaci a instalaci elektronických komponent.

Obr. 3.8: Robomaster S1 s vytvořeným rámcem pro elektroniku

4 Software

Jak bylo naznačeno v předchozí kapitole, část softwaru poběží na desce OpenCR a část na Jetsonu Nano. Hlavní logika programu je spuštěna na Jetsonu Nano, který na základě dat ze senzorů posílá požadavky ohledně akčního zásahu desce OpenCR. Deska OpenCR svým programem zajišťuje pravidelné posílání CAN instrukcí robotu Robomaster S1 a zároveň čte data ze senzorů, které posílá Jetsonu Nano na zpracování (mimo LiDAR, který je připojen přímo k Jetsonu Nano).

Kód pro pravidelné posílání CAN zpráv, které ovládají robota, je velmi důležitým softwarovým komponentem celé aplikace. Tento kód byl již částečně napsán v práci Jana Bělohlávka [7], ale protože původní kód byl psán pro mikroprocesorovou vývojovou platformu ESP32 [11], bylo třeba kód přepsat pro desku OpenCR. Zároveň došlo ke značnému přepracování kódu v rámci zpřehlednění a optimalizace.

V následující části budou představeny technologie použité při tvorbě softwaru a vysvětlena jejich funkce v kontextu aplikace.

4.1 ROS2 a MicroROS

Software vytvořený v rámci této diplomové práce využívá ROS2 [24] a MicroROS [28] k efektivní komunikaci a předání potřebných informací mezi jednotlivými částmi softwaru.

ROS2 je moderní open-source framework, který byl vytvořen pro vývoj robotických aplikací. Využívá distribuovanou komunikaci k propojení jednotlivých komponent. Klíčovou roli zde hraje DDS (Data Distribution Service), která umožňuje komunikaci mezi komponenty prostřednictvím publikování a odebírání zpráv nebo díky posílání požadavku na data.

Obr. 4.1: ROS logo

Pomocí ROS2 můžeme tedy rozdělit software do komponent. Každá komponenta obsahuje nějakou funkci robota. Tento modulární přístup umožňuje jednoduché rozšíření softwaru o již vytvořené komponenty, jako jsou například komponenty pro

publikování zpráv z různých senzorů nebo komponenta pro navigaci robota mezi překážkami.

Komponentu stačí spustit a zajistit, aby mohla přijímat potřebná data, maximálně pozměnit některá nastavení související se spuštěným algoritmem. Všechno ostatní už je zařízeno v rámci komponenty. Výsledky spuštěného programu publikuje komponenta na určité téma, to může odebírat jiná komponenta a s tímto výstupem pracovat.

Praktickým příkladem je LiDAR. Pro LiDAR pouze stáhneme kód komponenty zpracovávající data z LiDARu, spustíme a již můžeme ve zbytku aplikace pracovat s naměřenými daty.

Další velkou výhodou ROS2 je, že umožňuje real-time vizualizaci publikovaných dat pomocí programu Rviz [25] nebo simulaci v programu Gazebo [23].

Program Gazebo vytváří simulační prostředí, které zahrnuje realistickou fyziku, senzoriku a interakci 3D objektů. Tato simulace umožňuje testování robotických algoritmů a systémů v realistickém virtuálním prostředí. Pokud chceme posílat zprávy ve stylu ROS2 z desky OpenCR, potřebujeme MicroROS. MicroROS je moderní framework, který umožňuje integrovat technologii ROS do mikrokontrolérů. ROS2 se běžně používá ve výkonných výpočetních platformách, MicroROS přináší možnost využít myšlenky ROS2 na zařízeních s omezenými hardwarovými zdroji, jako jsou senzory, aktuátory nebo malé embedded systémy.

Obr. 4.2: MicroROS logo

Aby mohl ROS2 komunikovat s MicroROSem, musí být jednotlivé zprávy přeloženy. Překlad zpráv mezi ROS2 a MicroROSem zajišťuje MicroROS agent. Ten se stará o to, že publikované zprávy budou dostupné na obou stranách komunikačního kanálu.

4.2 Docker

Důležitou součástí vytvořeného softwaru je technologie Docker [15]. Docker slouží ke kontejnerizaci aplikace. Kontejnerizace aplikace extrahuje vytvořený software do izolovaného softwarového prostředí. Toto izolované softwarové prostředí se nazývá kontejner, jedná se o lehký a přenositelný balíček, který obsahuje všechny nezbytné

komponenty, jako jsou kód, knihovny a závislosti, aby aplikace mohla běžet konzistentně v různých prostředích. Tímto způsobem eliminuje problémy způsobené rozdíly v konfiguraci operačního systému nebo knihoven a poskytuje spolehlivé a opakovatelné nasazení softwaru.

Docker je tedy vhodný pro nasazení softwaru na různé robotické platformy a je zejména vhodný pro hejnové roboty, kdy je třeba aplikovat stejný software na vícero robotech. Docker v tomto případě zajistí jednoduché spuštění softwaru na jednotlivých robotech.

Obr. 4.3: Docker logo

Docker navíc disponuje nadstavbou Docker Compose, která umožňuje správu vícero kontejnerů zároveň. Pomocí nadstavby Docker Compose jsme schopni spustit více kontejnerů s různými nastaveními, jako je třeba nastavení sítě. Dále umožňuje definovat proměnné prostředí, závislosti mezi službami a automatizovat jejich nasazení pomocí jednoduchého YAML souboru.

4.3 Implementace

Algoritmus použitý pro pohyb ve formaci pro decentralizované hejnové roboty byl popsán v rešeršní části práce. Hlavní logika algoritmu vykonávaná na Jetsonu Nano je implementována zcela v duchu ROS2 technologie. To znamená, že vytvořený software je rozdělen do ROS2 balíčků, obsahujících jednotlivé komponenty.

4.3.1 Vývojové prostředí

Za zmínku stojí vývojové prostředí vytvořené pro implementaci softwaru. Řešení je postaveno na vývojové platformě Visual Studio Code [19], která nabízí rozšíření Dev Containers. Rozšíření Dev Containers využívá technologie Docker, popsané v předchozí kapitole.

Visual Studio Code spolu s rozšířením Dev Containers nám umožňuje vytvořit šablonu pro vývojové prostředí, ze které pak můžeme jednoduše vytvořit docker

kontejner, na který se můžeme vzdáleně připojit pomocí Visual Studio Code a vyvíjet program bez potřeby cokoli stahovat lokálně do našeho počítače.

Obr. 4.4: VS Code logo

Rozšíření Dev Containers navíc nabízí již mnoho předpřipravených šablon, které shromažďují veškeré programy a skripty potřebné pro vývoj určité technologie, v našem případě technologie ROS2.

Pomocí JSON souboru a souboru Dockerfile, které vytvoří rozšíření Dev Containers, jsme schopni nastavit různé detaily vývojového prostředí, jako například která další rozšíření stáhnout do kontejneru, umožnit přístup k připojeným zařízením na našem lokálním počítači a tak dále.

JSON spolu se souborem Dockerfile si následně může jakýkoli další vývojář stáhnout a pomocí rozšíření Dev Containers vytvořit identické vývojové prostředí bez nutnosti cokoliv lokálně stahovat.

V našem případě tak můžeme bezproblémově používat technologie ROS2, Rviz, Gazebo atd. s tím, že veškeré závislosti už jsou vyřešeny za nás a kdyby chtěl někdo na naši práci navázat, nemusí složitě stahovat veškeré programy a skripty, ale má vývojové prostředí již nachystané.

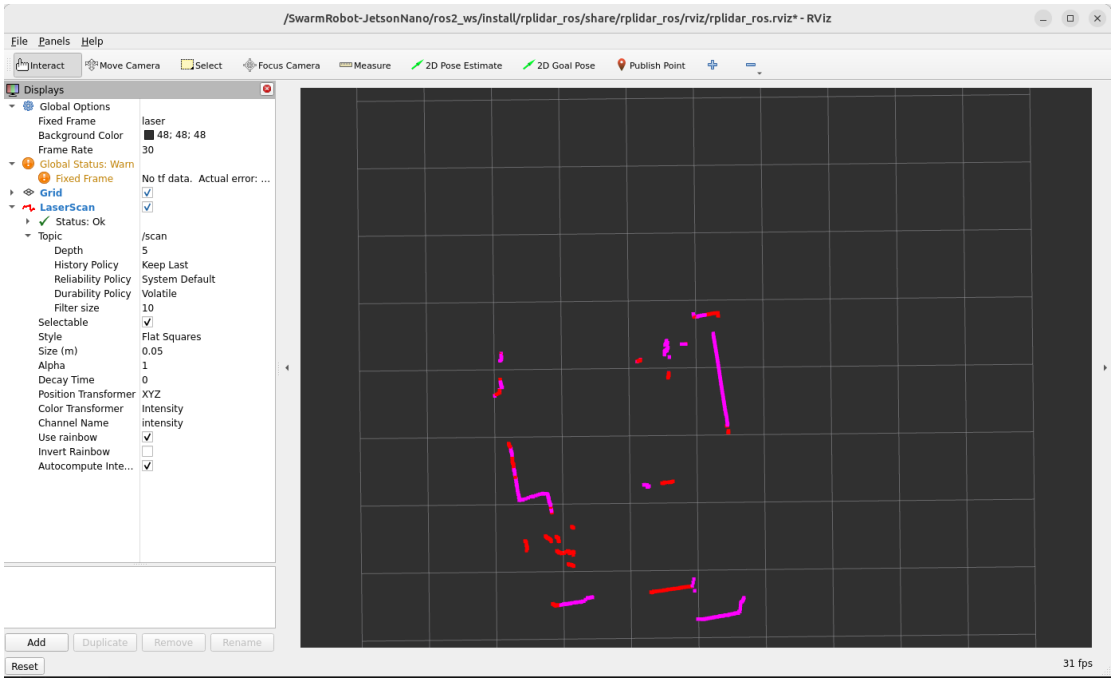
4.3.2 Čtení a interpretace dat z LiDARu

Pro aplikaci algoritmu pro pohyb ve formaci je třeba, aby měl každý robot k dispozici informace o vzdálenosti od sousedních robotů.

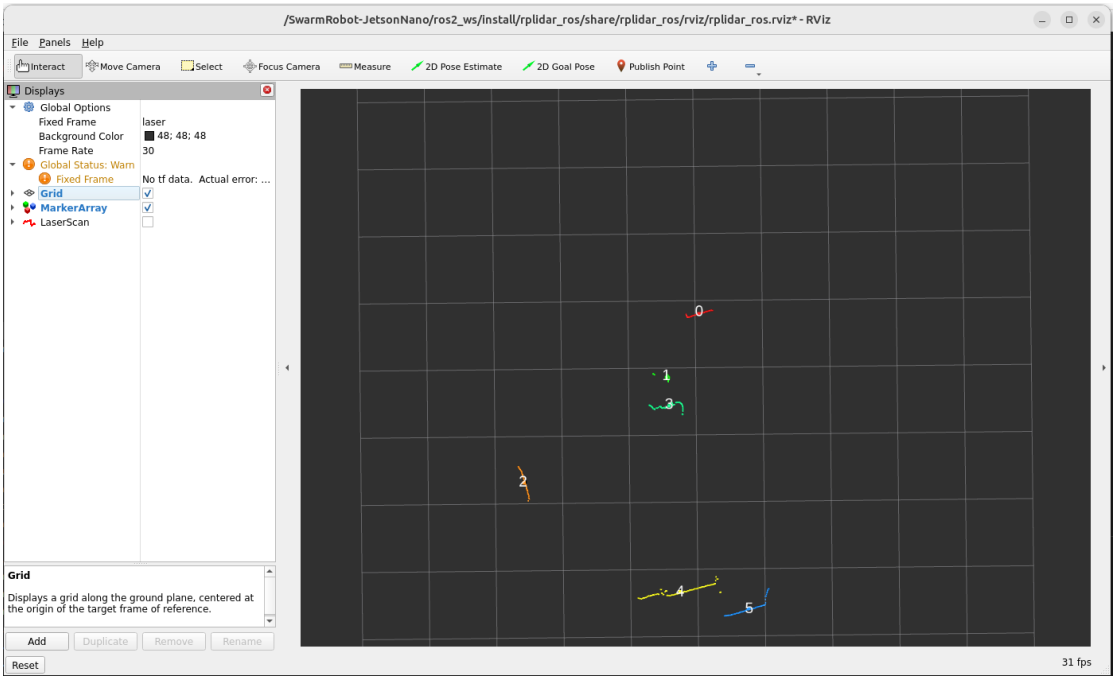
Sousední roboti jsou rozeznáni na základě dat z LiDARu, které jsou sdíleny a publikovány pomocí ROS2 balíčku s názvem `rplidar_ros` [13]. Tento balíček zprostředkoval přímo výrobce LiDARu, firma Slamtech.

Umožňuje propojení senzorů RPLIDAR A1, A2, A3, S1 a dalších s ROS2 ekosystémem, dále zajišťuje komunikaci se senzorem, publikování dat a základní konfiguraci.

K interpretaci dat z LiDARu byl použit ROS2 balíček `laser_segmentation` [12].



Obr. 4.5: Vizualizace dat z LiDARu



Obr. 4.6: Vizualizace segmentace dat z LiDARu

Autorem je Alberto Tudela. Tento balíček obsahuje komponentu, která data z LiDARu rozdělí do segmentů, a tyto segmenty následně publikuje.

Tyto segmenty pak následně reprezentují jednotlivé roboty. Bylo třeba nastavit parametry komponenty, tak aby detekované segmenty reprezentovaly potenciálního robota.

4.3.3 Sledovač

Jakmile máme detekované potenciální roboty, je třeba ověřit, zda se opravdu jedná o sousední roboty, nebo jen chybu balíčku či detekovaný předmět, který není robotem.

Pro tento účel byl vytvořen ROS2 balíček, který na základě detekovaných bodů a počátečních pozic těchto bodů sleduje pohyb robotů. Tím, že zná polohu robotů, dokáže rozlišit, které detekované body reprezentují sousedního robota a které ne.

Tento balíček, který se jmenuje `tracker`, využívá Kalmanův filtr s modelem konstantní rychlosti ke sledování bodů v prostoru. V případě detekce bodů pomocí LiDARu se jedná o sledování bodů v 2D, ale balíček je napsán tak, že je schopen sledovat body i ve 3D.

Matice pro Kalmanův filtr použitý ve sledovači byly odvozeny z článku [17].

Stavový vektor x Kalmanova filtru shromažďuje informace o poloze a rychlosti ve všech třech dimenzích.

$$x = \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \end{bmatrix}$$

Obr. 4.7: Stavový vektor x

Stavová matice A popisuje, jak se stavový vektor x mění mezi jednotlivými kroky. Matice vstupního řízení B určuje, jak ovlivňují vstupy systému (v tomto případě akcelerace) stav systému. Obě tyto matice vycházejí z modelu konstantní rychlosti. Proměnná Δt zaštiťuje dobu uplynulou mezi měřeními a je dána staticky jako vstupní parametr komponenty.

$$A = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Obr. 4.8: Stavová matice A.

$$B = \begin{bmatrix} \frac{1}{2}\Delta t^2 & 0 & 0 \\ 0 & \frac{1}{2}\Delta t^2 & 0 \\ 0 & 0 & \frac{1}{2}\Delta t^2 \\ \Delta t & 0 & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & \Delta t \end{bmatrix}$$

Obr. 4.9: Matice vstupního řízení B

Měřící matice H určuje výstup měření. Při sledování nás zajímá pouze poloha, proto vypadá matice následovně.

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Obr. 4.10: Měřící matice H

Matice Q je kovarianční maticí procesního šumu, reprezentuje nejistoty v modelu. Pro model konstantní rychlosti má následující tvar. Kde σ^2_{acc} je variancí procesního šumu, tedy nejistota modelu.

$$Q = \begin{bmatrix} \frac{\Delta t^4}{4} & 0 & 0 & \frac{\Delta t^3}{2} & 0 & 0 \\ 0 & \frac{\Delta t^4}{4} & 0 & 0 & \frac{\Delta t^3}{2} & 0 \\ 0 & 0 & \frac{\Delta t^4}{4} & 0 & 0 & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & 0 & 0 & \Delta t^2 & 0 & 0 \\ 0 & \frac{\Delta t^3}{2} & 0 & 0 & \Delta t^2 & 0 \\ 0 & 0 & \frac{\Delta t^3}{2} & 0 & 0 & \Delta t^2 \end{bmatrix} \cdot \sigma_{\text{acc}}^2$$

Obr. 4.11: Kovarianční matice procesního šumu Q

Naopak nejistotu měření, tedy senzorů, reprezentuje kovarianční matice R. Proměnná σ_{pos}^2 zde znázorňuje varianci měřícího šumu, tedy jak nepřesné jsou senzory.

$$R = \begin{bmatrix} \sigma_{\text{pos}}^2 & 0 & 0 \\ 0 & \sigma_{\text{pos}}^2 & 0 \\ 0 & 0 & \sigma_{\text{pos}}^2 \end{bmatrix}$$

Obr. 4.12: Kovarianční matice měřícího šumu R

Počáteční nejistotu odhadu udává kovariační matice P, která byla zvolena následovně.

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Obr. 4.13: Počáteční kovarianční matice P

Komponenta `tracker` pro sledování ostatních robotů je navržena tak, aby se dala znovu použít i pro sledování čehokoliv jiného. Navíc není závislá striktně na datech z LiDARu.

Pro tuto komponentu byl vytvořen další ROS2 balíček, který sdružuje typ zpráv pro komunikaci s `tracker` komponentou.

Tento typ zpráv obsahuje zprávu pro inicializaci sledovaných objektů, kdy každému sledovanému objektu je přiřazen identifikátor a počáteční poloha spolu s metadaty obsahujícími čas detekce a systém souřadnic, ve kterých byl detekován.

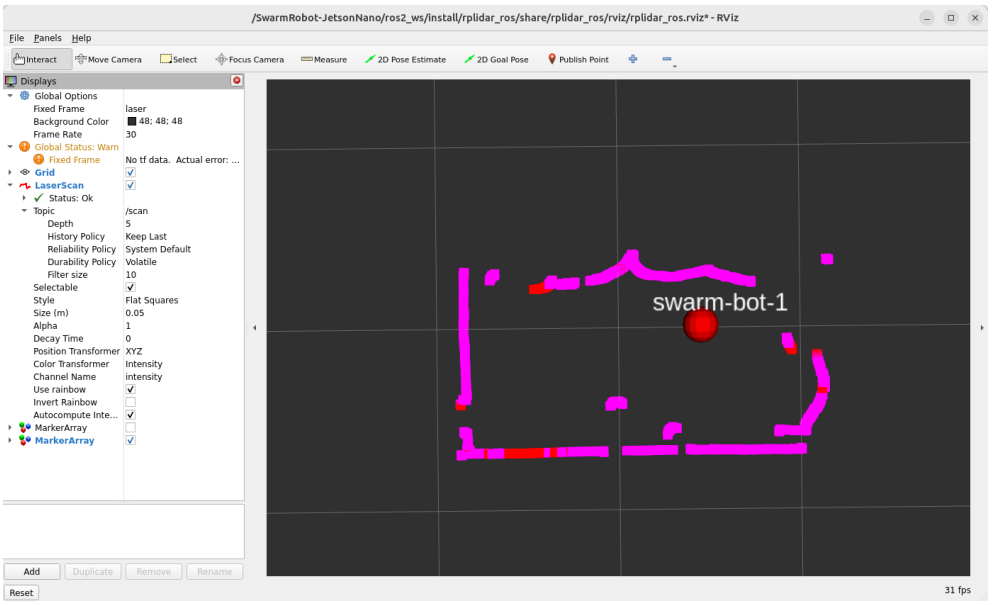
Další typ zprávy reprezentuje všechny detekované objekty, které jsou posílány komponentě `tracker` a na základě kterých komponenta provádí sledování.

Komponenta následně publikuje sledované objekty spolu s jejich identifikátorem a novou polohou.

Prakticky tedy může jakákoli jiná komponenta detekovat objekty, které chce sledovat, a to i jiným způsobem než pomocí LiDARu, a posílat je komponentě pro sledování.

Komponenta `tracker` tedy přijímá zprávy o možných polohách sledovaných předmětů (neřeší detekci), na základě kterých provede sledování a publikuje sledované předměty s jejich přiřazenými polohami.

Pro ladění komponenty a objevení různých chyb v programu publikuje komponenta navíc zprávy, které se dají jednoduše vizualizovat v programu Rviz. V programu Rviz reprezentuje sledovaný objekt sféra spolu s identifikátorem objektu, můžeme tak sledovat polohu sledovaného objektu v reálném čase.



Obr. 4.14: Vizualizace sledování objektu

4.3.4 Kontroler

Pro zpracování informací o sousedících robotech a jejich následnou interpretaci byla vytvořena komponenta `controller`.

Tato komponenta inicializuje sledování ostatních robotů, tím, že pošle inicializační zprávu s potřebnými informacemi komponentě `tracker`. Následně přijímá již segmentovaná data z LiDARu, zpracuje je a posílá je komponentě `tracker`, která jí odpoví zprávou obsahující aktualizované polohy sledovaných robotů.

Komponenta `controller` obsahuje dva regulátory. První regulátor je regulátor, který funguje na principu umělého potenciálového pole. Díky informacím o poloze sousedních robotů počítá potřebný akční zásah v podobě požadované rychlosti.

V rámci tohoto regulátoru je ještě implementovaný jednoduchý filtr v podobě dolní propusti prvního řádu. Tento filtr zajišťuje plynulý přechod jednotlivých akčních zásahů. Pro stabilitu řešení byl ještě přidán parametr s názvem *deadzone*, který vynuluje spočítaný akční zásah, pokud se robot nachází v určité vzdálenosti od požadované polohy.

Druhý regulátor byl přidán kvůli driftu, který robot vykazoval při řízení. Robot při zadání rychlosti sice jel správným směrem, ale postupně se začal čím dál více natáčet. Ukázalo se, že problém byl způsoben absencí regulátorů na motorech robota. Bylo tedy třeba aplikovat vlastní regulátor. Regulátor je klasický zpětnovazební PID regulátor, který udržuje požadovaný směr robota.

Robot sice disponuje enkódery, které mohou sloužit k poskytnutí zpětné vazby o natočení robota, ale tyto enkódery jsou součástí uzavřeného řešení robota a data z enkóderů nejsou přístupná. K získání zpětné vazby pro regulátor byl do řešení integrován ROS balíček `ros2_laser_scan_matcher`, který díky datům z LiDARu odhaduje pohyb robota, provádí tzv. odometrii. Odometrie získaná pomocí tohoto balíčku slouží jako zpětná vazba pro PID regulátor.

Akční zásah z obou regulátorů se složí dohromady a komponenta `controller` následně pošle zprávu s požadovanou rychlostí desce OpenCR, kterou, aby mohla být poslána na mikrokontrolér, přeloží MicroRos agent. Deska OpenCR následně zprávu zpracuje a pomocí sběrnice CAN pošle instrukce robotu Robomaster S1.

4.3.5 Spuštění programu na Jetsonu Nano

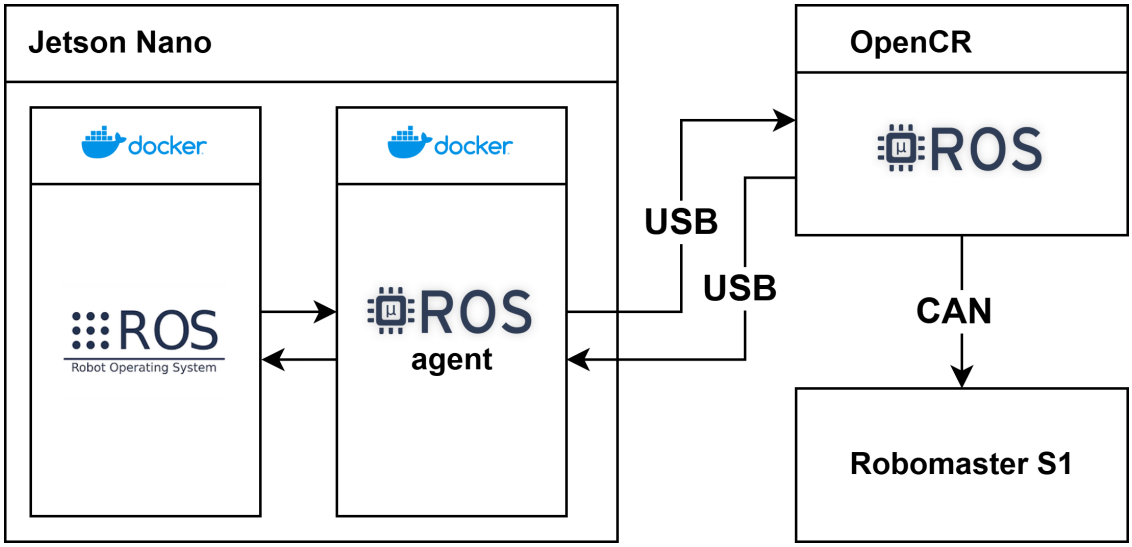
Pro efektivní spuštění jednotlivých komponent jako celku byl vytvořen další balíček s názvem `swarm_robot_launch`. Tento balíček sdružuje nastavení pro jednotlivé komponenty a využívá skriptů vytvořených pro spuštění těchto komponent ke spuštění komponent jako celku a spuštění tak celého ROS2 ekosystému.

Pro spuštění celého programu se všemi závislostmi slouží Docker Compose, který mimo vytvoření izolovaného prostředí se všemi závislostmi, sdružuje nastavení pro celé prostředí. Například komunikaci jednotlivých kontejnerů na lokální síti počítače nebo nastavení komunikace s připojeným hardwarem.

Pro spuštění celého ROS2 ekosystému včetně MicroROS agenta stačí stáhnout kód na Jetson Nano, zkompilovat ho pomocí příkazu: `docker-compose build` a vše spustit pomocí příkazu `docker-compose up`. Díky Dockeru není třeba již nic pro běh programu stahovat nebo nastavovat.

4.3.6 MicroROS komponenta a odesílání instrukcí

Na desce OpenCR je implementovaná MicroROS komponenta, která zajišťuje přijetí instrukcí z ROS2 ekosystému spuštěného na Jetsonu Nano a následné poslání zakomponování daných instrukcí do rozluštěných CAN zpráv a následné poslání zpráv přes CAN sběrnici na Robomastera S1. Jak již bylo řečeno, překlad zpráv mezi ROS2 a MicroROS ekosystémem zajišťuje MicroROS agent. Zapojení systému a jednotlivé závislosti jsou přehledně zobrazeny na následujícím obrázku.



Obr. 4.15: Software zapojení - přehled

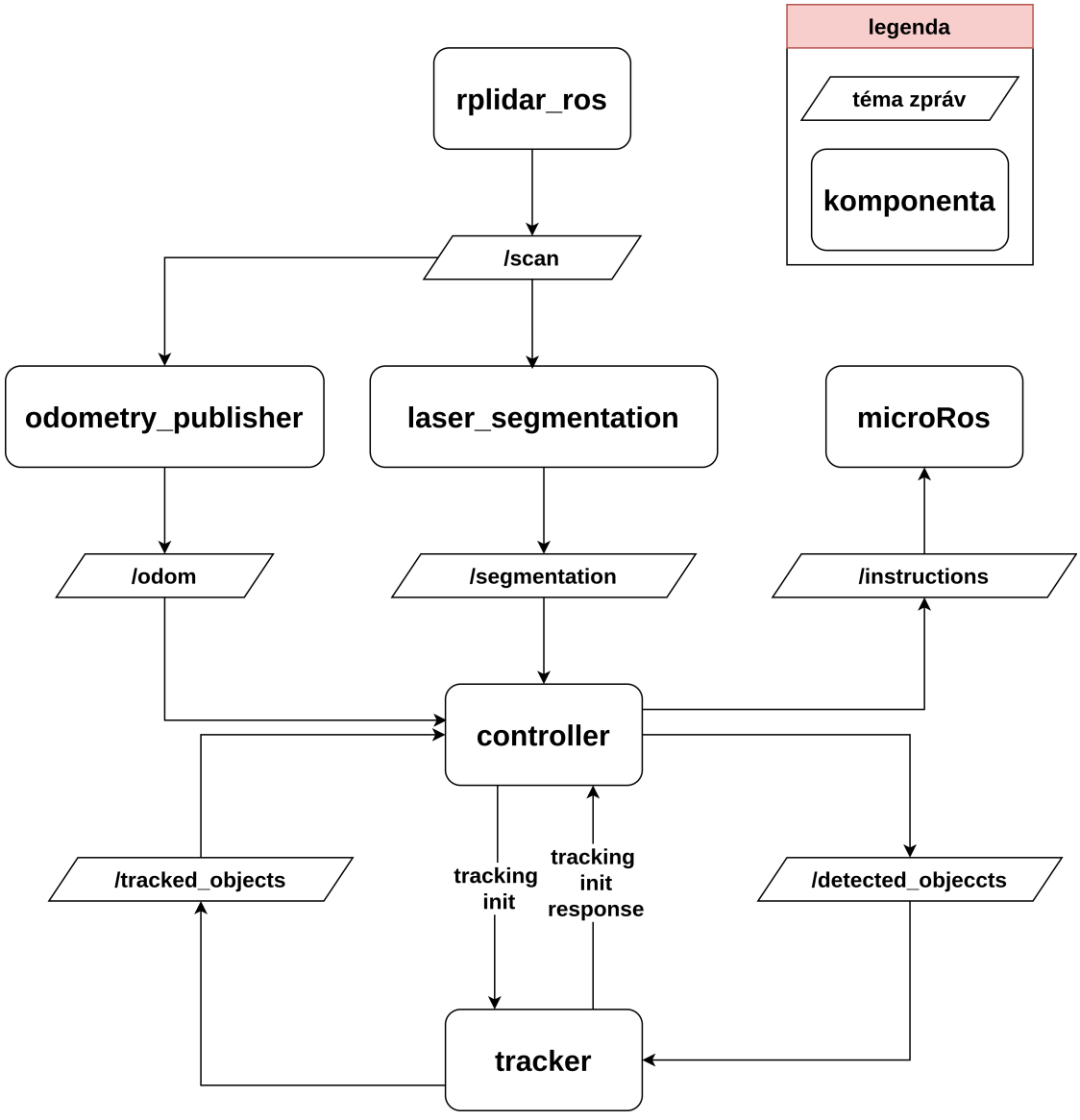
CAN zprávy s integrovanými instrukcemi jsou na Robomaster S1 robota posílány v pravidelných intervalech. Zprávy je třeba podle typu poslat každých 10, 100, 1000 milisekund. Toho je docíleno tím, že jsou jednotlivé zprávy vždy v časových intervalech nahrávány do kruhového bufferu, odkud jsou následně pomocí CAN sběrnice poslány na robota.

Před nahráním do kruhového bufferu jsou rozluštěné zprávy zpracovány, dochází k výpočtu kontrolního součtu a integraci instrukcí. MicroROS komponenta čeká na přijetí zprávy o změně hodnoty instrukcí, pokud dojde ke změně, zapíše novou hodnotu do bufferu dedikovanému pro uložení instrukcí. Z tohoto bufferu si následně program vyčítá hodnotu při zpracování rozluštěných zpráv.

V případě detekce chyby v logice MicroROS komponenty začne program signalizovat tuto událost blikáním červené LED diody. Pokud není deska připojena k MicroROS agentu, bliká zelená LED dioda, která zůstane svítit zeleně, jakmile se MicroROS agent úspěšně připojí.

4.3.7 Shrnutí

Program pro decentralizovaný pohyb ve formaci je rozdělen na dvě výpočetní jednotky: jednodeskový počítač Jetson Nano a desku OpenCR.



Obr. 4.16: ROS2 ekosystém - přehled

Část programu na Jetsonu Nano se, díky zprostředkovaným datům z LiDARu, stará o sledování pozic okolních robotů relativně vůči vlastní pozici, dále díky informacím o poloze sousedních robotů počítá potřebný akční zásah k udržení formace. Pro regulaci natočení je implementován PID regulátor se zpětnou vazbou v podobě odometrie z LiDARu. Informace o akčním zásahu posílá na desku OpenCR.

Část programu na desce OpenCR v pravidelných intervalech posílá zpracované instrukce přes sběrnici CAN na robota Robomaster S1.

K nasazení programu na Jetson Nano byla využita technologie Docker, která umožňuje vytvořit reprodukovatelné prostředí pro nasazení kódu.

Program je psán zcela v duchu ROS2 technologie. To znamená, že jednotlivé části programu jsou implementovány jako ROS2 komponenty. ROS2 ekosystém na desce OpenCR nahrazuje technologie MicroROS, která komunikuje s ROS2 ekosystémem na Jetsonu Nano pomocí MicroROS agenta.

Jednotlivé komponenty spolu komunikují pomocí zpráv publikovaných na různá témata. Jednotlivá témata celého ROS2 ekosystému jsou přehledně zobrazená na diagramu 4.16.

5 Zhodnocení

Diplomová práce se zabývala rešerší problematiky hejnových algoritmů, ověřením vybraných algoritmů ve vytvořené simulaci a následným ověřením algoritmu pro pohyb ve formaci na reálné robotické platformě vytvořené v rámci diplomové práce.

V rámci simulace byly ověřeny algoritmy pro pohyb ve formaci, které se lišily především implementací umělých virtuálních sil. Dále dva algoritmy pro optimalizovaný sběr potravy a algoritmus pro uspořádání robotů do linie. Na robotické platformě vytvořené v rámci této diplomové práce byl aplikován vybraný algoritmus pro decentralizovaný pohyb robotů ve formaci.

Algoritmus vybraný pro praktický pokus na vytvořené robotické platformě byl vybrán na základě výstupů ze simulace.

V simulaci vykazoval nejlepší stabilitu při pohybu ve formaci z předem zkonfigurované formace dynamický model, který zároveň vykazoval nejlepší výsledky, co se týče uspořádání do formace a až následném pohybu.

Model s měkkým potenciálem vykazoval obdobně stabilní výsledky. Výsledky simulací jednotlivých algoritmů byly značně závislé na simulačním kroku, který v podstatě simuloval rychlost odezvy jednotlivých senzorů. Což bylo nejspíš způsobeno velkým nárůstem akčního zásahu v případě přiblížení robotů během měření na vzdálenost bližší než vzdálenost požadovanou.

Pro praktickou implementaci byl nakonec zvolen model s měkkým potenciálem. Důvodem pro upřednostnění tohoto modelu nad dynamickým modelem, který vykazoval obdobné výsledky, byl fakt, že dynamický model potřebuje k určení potřebného akčního zásahu informaci o relativní rychlosti ostatních robotů. Tuto rychlost lze získat pomocí derivace relativní polohy sousedních robotů. Derivace se ale v praktické implementaci často nepoužívá a to z důvodu šumu na senzorech, který může způsobit chybný numerický výpočet a tím i chybný výpočet akčního zásahu, který by způsobil nestabilitu řešení.

Zvolený algoritmus při praktických testech potvrdil výsledky dosažené v simulaci. Jediným problémem daného řešení jsou horší regulační vlastnosti, pokud se k sobě roboti přiblíží na vzdálenost menší než mezní vzdálenost algoritmu. Tento problém je způsoben vlastnostmi potenciálového pole, které nedokáže tak dobře zasáhnout při přiblížení robotů k sobě.

Závěr

V úvodu diplomové práce je provedena důkladná rešerše, která představuje hejnové algoritmy se zaměřením na agregaci, prohledávání prostoru a sběr potravy, pokrytí prostoru, rozdělení úkolů a další. Vybrané algoritmy představené v rešeršní části jsou ověřeny na jednoduchém hejnovém 2D simulátoru vytvořeném k základnímu ověření hejnových algoritmů a jejich vizualizaci.

Byla také provedena rešerše stávajících řešení hardwaru hejnových robotů, která byla brána v potaz při tvorbě hardwarové nadstavby robota Robomaster S1. Byl vytvořen rámec pro elektroniku, který byl osazen hlavní výpočetní jednotkou Jetsonem Nano, napájený DC/DC konvertorem přímo z robota, deskou OpenCR a LiDARem.

Na vytvořené robotické platformě byl ověřen algoritmus pro decentralizovaný pohyb ve formaci. Pro měření vzdálenosti mezi roboty byl vybrán LiDAR, jelikož do značné míry eliminuje problém přeslechů, který například znemožňuje použít podstatně levnější ultrazvukový senzor.

Vytvořený software pro aplikaci algoritmu pohybu ve formaci je psán převážně v C/C++ a pythonu, využívá ROS2 a MicroROS ke čtení a zpracování dat ze senzorů a následné komunikaci mezi jednotlivými částmi hardwaru. Software využívá k pohybu ve formaci dva regulátory. První, APF regulátor, reguluje vzdálenost od ostatních robotů pomocí umělého potenciálního pole. Druhý, PID regulátor, který udržuje požadované natočení. PID regulátor bylo třeba přidat, jelikož podvozek robota Robomaster S1 nedisponuje regulátorem a robot při aplikaci akčního zásahu vykazoval drift. Navíc byl vytvořen sledovač, který slouží k určení polohy ostatních robotů na základě dat z LiDARu.

K nasazení softwaru na jednotlivé hejnové roboty byla využita technologie Docker, která se perfektně hodí pro problematiku hejn, kde je třeba nasadit stejný software na vícero robotech.

V rámci diplomové práce bylo vytvořeno hejno tří robotů, na které byl nasazen software, který při praktických testech prokázal svou spolehlivost při řešení zadané úlohy. Hejnoví roboti jsou schopni se přesouvat ve formaci linie a šípu.

Vytvořená robotická platforma a softwarové řešení, které je psáno tak, že je velmi lehce rozšiřitelné a přepoužitelné, může v budoucnu sloužit k ověření dalších hejnových algoritmů anebo i k implementaci úloh nehejnových, jako třeba detekce překážek. Zároveň byla vytvořena komunikace mezi jednotlivými roboty přes Wi-Fi, která může být využita k vytvoření centralizovaných algoritmů pro řízení hejna anebo k vytvoření programu pro hromadné nahrání softwaru.

Literatura

[1] NASA Swarmathon — swarmathon.cs.unm.edu. https://swarmathon.cs.unm.edu/, [29-11-2024].

[2] RoboMaster S1 - DJI — dji.com. https://www.dji.com/cz/robomaster-s1, [29-11-2024].

[3] Swarm-bots project — swarm-bots.org. https://www.swarm-bots.org/, [29-11-2024].

[4] Swarmanoid project — swarmanoid.org. https://www.swarmanoid.org/, [29-11-2024].

[5] Bahaidarah, M.; Marjanovic, O.; Rekabi-Bana, F.; aj.: Improving Formation in Swarm Robotics with a Leader-Follower Approach. In *2024 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2024, s. 1447–1452, doi: 10.1109/ICMA61710.2024.10633130.

[6] Brutschy, A.; Pini, G.; Pinciroli, C.; aj.: Self-organized task allocation to sequentially interdependent tasks in swarm robotics. *Autonomous Agents and Multi-Agent Systems*, roèník 28, 01 2014: s. 101–125, doi:10.1007/s10458-012-9212-y.

[7] Bělohlávek, J.: *Vytvoření nehomogenního hejna robotů*. Bakalářská práce, Vysoké učení technické v Brně. Fakulta strojního inženýrství. Ústav mechaniky těles, mechatroniky a biomechaniky, 2024.
URL https://hdl.handle.net/11012/248354

[8] Christensen, A.; O'Grady, R.; Dorigo, M.: From Fireflies to Fault-Tolerant Swarms of Robots. *Evolutionary Computation, IEEE Transactions on*, roèník 13, 09 2009: s. 754 – 766, doi:10.1109/TEVC.2009.2017516.

[9] Dorigo, M.: SWARM-BOT: an experiment in swarm robotics. In *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, 2005, s. 192–200, doi:10.1109/SIS.2005.1501622.

[10] Drátek.cz: XL4015.
URL https://dratek.cz/arduino/121946-step-down-modul-5a-xl4015-dc-dc-cc-cv.html

[11] espressif: esp32.
URL https://www.espressif.com/en/products/socs/esp32

[12] GitHub, I.: Laser segmentation balíček.
URL https://github.com/ajtudela/laser_segmentation/tree/humble

[13] GitHub, I.: RPLiDAR ROS balíček.
URL https://github.com/Slamtec/rplidar_ros/tree/ros2

[14] Gonçalves, P.; Torres, P.; Alves, C.; aj.: The e-puck, a Robot Designed for Education in Engineering. *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, roèník 1, 01 2009.

[15] Inc., D.: Docker.
URL https://www.docker.com/

[16] Jeong, D.; Lee, K.: Dispersion and Line Formation in Artificial Swarm Intelligence. 06 2014.

[17] Khobahi, S.: Object tracking using kalman filter. *github. com*, 2017.

[18] Lv, Y.; Yang, X.; Yang, Y.; aj.: Formation control of UAVs based on artificial potential field. *MATEC Web of Conferences*, roèník 189, 01 2018: str. 10018, doi:10.1051/matecconf/201818910018.

[19] Microsoft: Visual Studio Code. https://code.visualstudio.com/, 2024, accessed: 2025-02-07.

[20] Na, S.; Krajník, T.; Lennox, B.; aj.: Federated Reinforcement Learning for Collective Navigation of Robotic Swarms. 02 2022, doi:10.48550/arXiv.2202.01141.

[21] Nauta, J.; Van Havermaet, S.; Simoens, P.; aj.: Enhanced foraging in robot swarms using collective Lévy walks. 11 2020, doi:10.3233/FAIA200090.

[22] Rivera, A.: A generalized solution to the coverage problem in swarm robotics: The force vector algorithm. In *2016 IEEE MIT Undergraduate Research Technology Conference (URTC)*, 2016, s. 1–4, doi:10.1109/URTC.2016.8284072.

[23] Robotics, O.: Gazebo.
URL https://gazebosim.org/home

[24] Robotics, O.: ROS Humble.
URL https://docs.ros.org/en/humble/index.html

[25] Robotics, O.: Rviz.
URL https://docs.ros.org/en/humble/Tutorials/Intermediate/RViz/RViz-User-Guide/RViz-User-Guide.html

[26] Robotis: Robotis e-Manual OpenCR.
URL https://emanual.robotis.com/docs/en/parts/controller/opencr10/

[27] Robotis: Robotis e-Manual Turtlebot3.
URL https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

[28] micro ROS: Micro-ROS.
URL https://micro.ros.org/docs/overview/features/

[29] Rosca, S.; Leba, M.; Sibisanu, R.: *SSVEP Based BCI Control of a Robot Swarm.* 05 2022, ISBN 978-3-031-04825-8, s. 296–305, doi:10.1007/978-3-031-04826-5_29.

[30] RPIshop.cz: NVIDIA Jetson Nano Developer Kit, verze B01, originál.
URL https://rpishop.cz/jetson-nano/1636-nvidia-jetson-nano-developer-kit-verze-b01.html

[31] Son, J.-H.; Ahn, H.-S.; Cha, J.: Lennard-jones potential field-based swarm systems for aggregation and obstacle avoidance. In *2017 17th International Conference on Control, Automation and Systems (ICCAS)*, 2017, s. 1068–1072, doi:10.23919/ICCAS.2017.8204374.

[32] Sutantyo, D.; Möslinger, C.; Read, M.; aj.: Collective-adaptive Levy-flight for underwater multi-robot exploration. In *2013 IEEE International Conference on Mechatronics and Automation, IEEE ICMA 2013*, 08 2013, doi:10.1109/ICMA.2013.6617961.

[33] Wenger, I.; Ebel, H.; Eberhard, P.: Anomalously acting agents: the deployment problem. 10 2023, doi:10.21203/rs.3.rs-3527368/v1.

[34] Yang, Y.; Zhou, C.; Tian, Y.: Swarm robots task allocation based on response threshold model. In *2009 4th International Conference on Autonomous Robots and Agents*, 2009, s. 171–176, doi:10.1109/ICARA.2000.4803959.

Seznam symbolů a zkratek

USB Universal Serial Bus

CAN Controller Area Network

CLW Collective Walk

ACLW Adaptive Collective Walk

VS Code Visual Studio Code

LiDAR Light Detection and Ranging

OpenCR Open-source Control Robot

ROS2 Robot Operating System 2

MicroROS Micro Robot Operating System

JSON JavaScript Object Notation

A Obsah elektronické přílohy

Elektronické přílohy pro tuto diplomovou práci jsou rozděleny do čtyř složek. Složka `Swarm` obsahuje implementaci hejnového simulátoru spolu s ukázkami jednotlivých hejnových algoritmů.

Složka `SwarmRobot-OpenCR` obsahuje kód, který, jak je zřejmé z názvu, běží na desce OpenCR. Tedy pravidelné přijímání a odesílání instrukcí.

Ve složce `SwarmRobot-JetsonNano` se vyskytuje kód spouštěný v rámci řešení na Jetsonu Nano. Tedy vytvoření izolovaného prostředí, sledování ostatních robotů a výpočet akčního zásahu.

Poslední složka obsahuje modely jednotlivých dílů vytvořeného rámce pro elektroniku.